A GENETIC ALGORITHM TO SOLVE A MULTI-PRODUCT DISTRIBUTION PROBLEM

by

Bruno Miguel Ribeiro Crétu

Master in Data Analytics Dissertation

Supervised by:

Professor Dalila B. M. M. Fontes

Faculdade de Economia

Universidade do Porto

2017

Biographical Note

Bruno Miguel Ribeiro Crétu was born on the 12th of May 1994, in the city of Porto. After finishing his bachelor in Business Administration at Porto School of Economics in 2015, he proceeded his studies with the Master in Data Analytics. During his bachelor, he became a member of a student organization, the Experience Upgrade Program, and was a summer intern at Frederico Mendes & Associados, a consulting firm.

While in his master he was admitted as an intern in Sonae's SR fashion division MO, fulfilling the role of Business Analyst. As of April 2017, he left his position to dedicate his time to the master thesis and other personal projects.

Additionally, he practices sport, namely rowing, since he was 9 years old. During this period, he competed internationally, for his club and national team.

Acknowledgements

First, I want to thank Professor Dalila for accepting this challenge and for her crucial help in building this thesis. Without her help, ideas and high spirits this wouldn't be possible.

A big thanks to all my family and friends. They listened to my complaints and understood my absence of mind during this period. Even though no one understood what I was stressing about, they took the time to listen and calm me in a certain way. To the family specifically, thank you for teaching me perseverance and ambition, it was key on hard moments. To the friends, thank you for the company, the drinks, the conversations that kept my mind of problems when I most needed.

As a rower, rowing was also the perfect escape to either get my mind of things or to think about how to overcome the issues that I came across. Thank you to the big team of Sport Clube do Porto for creating the perfect training atmosphere, and of course to my teammates Tito, João and Guilherme.

Finally, but not least important, to all the people of online forums or websites that posted questions and answers to all kinds of problems. For a person who is learning, many questions came up, but none was left unanswered. I know you will never get to read this, but a big thank you!

Resumo

Esta dissertação tem como objetivo criar uma ferramenta que apoie a tomada de decisão dos gestores de downstream de uma empresa de moda portuguesa. Os gestores devem decidir enviar determinados produtos de um armazém para cada uma das lojas e especificar as quantidades. Escolhemos abordar este problema duma perspetiva de minimização de custos, mas com especial interesse nas restrições.

Sendo um problema combinatório difícil, é necessário utilizar uma heurística. Como os Algoritmos Genéticos são conhecidos pela sua eficiência na resolução deste tipo de problemas, propomos construir um adequado ao problema a resolver. Os resultados mostram que o algoritmo é capaz de resolver o problema de forma eficiente, eficaz e, sobretudo, ser uma ferramenta de apoio à decisão.

Palavras chave: algoritmo genérico, gestor downstream, cadeia de abastecimento, otimização, moda.

Abstract

The dissertation aims at creating a tool to support the decision making of downstream managers of a portuguese fashion company. The managers must decide whether to send a certain product to each store, and specify the quantities, from one warehouse. We chose to approach the problem with a cost minimization perspective, but with special interest on the constraints.

Being a hard combinatorial problem, a heuristic had to be used. Since Genetic Algorithms have been known to be efficient at solving this kind of problems we propose to build one suited for it. The results will show that the algorithm is capable of solving the problem efficiently, effectively and, ultimately, being a decision support tool.

Keywords: genetic algorithm, downstream manager, supply chain, optimization, fashion.

Contents

List of Figures

List of Equations

List of Tables

1. Introduction

1.1. Objectives and motivation

The apparel and textile industry yields great importance to the European Union, employing more than 1.6 million people, in 2015[1]. Besides its importance to the job market, it is responsible for dressing people around the world which affects cultural interactions and standards.

The increasing volatility, competitiveness and short product lifecycle requires an efficient response from the market players (Bruce, Daly, & Towers, 2004). Depending on the strategic orientation of each company (e.g., luxury vs fast fashion), seeking lower costs to achieve high return on margin is a priority for the companies. To accomplish this, companies resort to global sourcing, adopting an integrated system known as supply chain management.

The tool that we propose to solve is aimed at supporting the decision making process of managers that integrate the supply chain of a fashion company. We aim at creating a tool that will effectively and efficiently give the quantities, of each product, that the manager should send to each store of the company, at a minimum cost. However, as it will be explained, cost is not the only issue, so here we are also proposing a new point-of-view of how to manage the stocks. The company has one warehouse that is stocked every week. With the stock available in the warehouse, the managers need to find the best way to allocate it to each store. Therefore, the algorithm's output will be the quantity of each product to be sent to each store, while minimizing the cost.

The problem addressed here belongs to the combinatorial class as it involves finding an assignment of a discrete finite set of objects, while satisfying a given set of conditions. In addition, realistic problem instances involve many objects and decisions. Therefore, the methodology to develop must be heuristic in nature. Amongst the many existing heuristics, genetic algorithms have been chosen, mainly due to:

[1] Statista: https://www.statista.com/statistics/417750/eu-european-union-textile-clothing-industry-employment/

i. The good performance (runtime, feasible solutions ratio and solutions quality) in many other combinatorial problems

ii. Mentor's experience with these types of heuristics, which will ease the learning process.

1.1. Methodology

We will use a genetic algorithm (GA) built from scratch in Python language. The programming language chosen is, first, the one that the mentee has some knowledge of and, secondly, Python's flexibility, simple syntax and massive community make it a perfect language to learn in such a short span. Flexible, because there are no hard rules on how to build a script and problems can be approached through many methods. The code is read like English, focuses on concepts and avoids details. Across the top programming communities (e.g., GitHub, StackOverFlow, Meetup, etc.) the language is on the top in terms of members and projects. This allows an easier option to solve any kind to problem.

The algorithm is basic in its implementation and contains the following usual features of a GA:

- Randomly generated population where each store and product is represented;
- Fitness function that computes the score of each individual solution not only based on the cost but also on the compliance with the constraints;
- Elitism that transfers the top percentage of solutions into the next generation;
- A crossover operator that pairs two chromosomes with an adaptation of the 1-point crossover and an elitist pool to select the parents;
- And finally the introduction of mutants instead of a mutation operator based on the biased random-key genetic algorithm (BRKGA) (Toso & Resende, 2015).

The code presented here was designed to answer to the problem described in the dissertation. However, with small changes to the code it may be adapted to similar problems, even if unrelated.

1.2. Context

To begin with, we need to understand the context where the supply chain is inserted, i.e., organizational structure and product structure. The problem that we tackle is based on a specific structure of a Portuguese Fashion Store. The information will not reveal any business specific aspects, the latter ones are omitted. Data deemed relevant, but impossible to bring to light, will be fictitious and properly stated in the assumptions.

Structurally speaking there are three main areas on the supply chain: Upstream, Downstream, and Quality Management. The tool being presented here is intended to be applied by the Downstream Manager (DM)

The DM's job is to determine the optimal quantity of each product to be sent to each store while satisfying certain business parameters. The parameters include minimum and maximum amounts per product per store. These limits are adjusted over time, mainly as a consequence of problems that the stores communicate to the central offices or as strategic decisions.

Currently, the decisions of whether to send each product or not is heavily dependent on appropriated days of coverage (DC). There is a business reference to how many DC there should be. The company has a threshold of 4 weeks. The DC provides the information of how many days the current in-store stock can cover, given the value/quantity of sales, over a certain period of time, as given in the following equation.

$$DC_i = \left(\frac{s_i}{\sum_{j=i}^{t} p_j}\right) \times t - i + 1$$

Equation 1 - Days of coverage formula.

The days of coverage in period *i* (DC_i) is calculated as the ratio of the stock in period *i* (s_i) and the to be projected sales (p) from period *i* until period *t*, where *t* is the size of the time window of analysis, usually the whole season (6 to 8 months).

However, currently, the sales used to obtain the coverage days are last year's sales plus a growth rate, based on the previous two weeks uplift. Using the previous year sales data as a decision factor to supply the stores raises the crucial issue: strategy. In other words, an apparel collection is built based on the homologous period (i.e., Autumn/Winter'17 is

based on Autumn/Winter'16), but the merchandiser[2] takes strategic decisions, not only to amend past failures but also to react to trends.

Consider the following example of how it impacts the DM's job: if the merchandiser plans to buy more shirts because the market points in that direction. The result of the DM's analysis will be high (unusual) DCs. The increased purchase of such items leads to an inventory that is larger than required to cover the last year's sales. If the DM is not aware of the strategic decision, he/she can make the decision of not sending more shirts so that stores are not overburdened with stock. The reverse may also happen. The DM might detect that there is not enough stock to supply all the stores. When the merchandiser proposedly bought less of that product to make up for waging on other products.

To overcome this, a budget that reflects the strategy of each type of product (e.g., T-shirts) is necessary. The DM needs to review the given initial budget, every week, alongside with a markets team to properly address outliers that are not obvious in the initial build of the budget. Or if any major changes are made to the promotional activities (i.e., discounts or store events).

The collection is split into two major groups that require different treatment. On the one hand, we have the seasonal collection. It changes according to the season and is rarely repeated after one season. On the other hand, there are items that are permanently in-store, which are known as the never out of stock products (NoS). They require a type of management different than that of the seasonal items.

The product structure is straightforward:

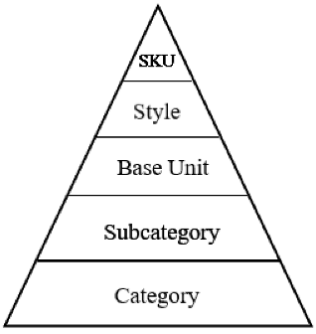


Figure 1 - Product Structure.

[2] Person responsible for planning the collection, who selects which products will be in-store, price and promotional highlight.

The category is the most macro point-of-view one can have at the company. At this level, there are eight (8) categories, four for woman and four for man, as follows: Apparel, Underwear/Nightwear, Footwear and Accessories. The Woman and Man Apparel are the main categories so each requires a DM. The other categories have one manager per gender (i.e., a DM for Underwear/Nightwear, Footwear, and Accessories for Man and another for Woman).

The subcategories aggregate the collection into segments. The aggregator is particularly important on the apparel and underwear/nightwear categories. The apparel subcategories correspond to brands (e.g., Casual Collection, Teen Collection and Basics Collection), while the underwear/nightwear are more specific groups (e.g., Nightwear, Underwear, Lingerie, Slippers, Socks).

The base unit identifies the type of product of each sub-category. One base unit can have multiple styles but it belongs to one subcategory only, examples of a base unit can be T-shirts or Outerwear.

The two most detailed levels are the Style and SKU. The Style is a specific product/color, so a red T-shirt and a green T-shirt are by default categorized under different styles each. There is never the same code season after season. Whereas, the SKU is the size of a given Style.

The supply is mainly done in packs of products. The packs have a set of sizes (e.g. one pack may contain 1 unit of size S, 2 of size M, 2 of size L, and 1 of size XL) and are used to do the first supply of the store. It represents 60-80% of the purchase done. Each store has a minimum pack required (associated with its sales and sales space capacity). If the sales are higher than expected, the downstream manager can re-supply using the "color-size". The "color-size" are specific sizes of a style (e.g., red T-shirt of size M).

Due to the data availability, we can only approach the problem from a category point of view. Thus, we will work with 8 products - Apparel, Underwear/Nightwear, Footwear and Accessories for both Man and Woman. Henceforth, when we refer to "product" we are talking about a specific category.

1.3. Dissertation structure

We will start by defining the problem in Chapter 2, which includes a detailed description of the problem, assumptions, variables, and the mathematical formulation. Chapter 3 will dive into the approach taken in solving the problem, and describe the construction of the algorithm. In Chapter 4, we will discuss the computational experiments and results with the methodology that is currently adopted. Finally, Chapter 5 will summarize the conclusions of the work developed and suggest further work and improvements.

2. Problem description

The problem to be solved in the dissertation fits, in most part, in the Lot Sizing Problem (LSP). However, as we will see, it includes characteristics of multiple variants of the latter one. The LSP has many variants resulting from the introduction of new issues to better describe the problem that each author wanted to tackle. Over time several extensions have been considered, such as, limited capacity of both production and storage (Rogers, 1958); Hanssmann, 1962, production of multiple items (Elmaghraby, 1978; Hsu, 1983; Du Merle, Goffin, Trouiller, & Vial, 2000) and backlogging (Zangwill, 1969). The combination of the modifications over time lead to the economic lot scheduling problem, which in addition to finding the production amount for each product one also wants to find out which product is to be produced and when.

In short, the LSP consists of determining the production quantities for a specific time horizon, on a single or multi-level, for a specific number of products. There are capacity and/or resource constraints, and it need to satisfy a known or unknown demand at a minimum cost while maximizing the service level.

This dissertation focuses on a specific context and uses a genetic algorithm to solve it. Nevertheless, it takes into consideration the inputs of several authors that already explored the problem, even if in different contexts. As we will see, the problem has been tackled from an industrial point of view, so for example, variables such as set up costs and machine scheduling have no importance in our problem.

The first LSP solution was introduced by Wagner and Whitin (1958). The authors designed an algorithm to solve the uncapacitated single lot sizing problem (USILP) optimally. The USILP has no bounds on production or inventory and no backlogging. The algorithm, named after both authors (Wagner Whitin algorithm), specifies the amount needed to satisfy a known demand over multiple, finite, periods. An unlimited fixed amount may be ordered, linear production and holding costs are considered. The algorithm efficiency was later improved by several authors (Federgruen & Tzur, 1991; Aggarwal & Park, 1993) and its computational complexity has been reduced from $O(n^2)$ to $O(n \log n)$. Although the WW algorithm is relevant, it cannot be applied to the problem we are trying to solve. The authors assume unlimited capacity to satisfy the given demand.

Several reviews (Wagner & Whitin, 1958; Kaminsky & Simchi-Levi, 2003; Karimi, Ghomi, & Wilson, 2003; Ben-Daya, Darwish, & Ertogral, 2008) of the lot sizing problem and its variants have been repeated over the years. Furthermore, the authors explore the models used to solve the problem, which we will discuss when introducing the solution approach chapter (Chapter 3).

Here we have a weekly stocked single warehouse that supplies all the stores of the company. The managers have to decide how much of each product to send to each store. The warehouse and stores have capacity limits that should not be exceeded[3]. In addition, there are transport quotas that must be respected. A minimum and maximum amount of product to send, to so that the store has quantities of every product in store.

The manager has a detailed, weekly, budget for each store, ideally, the stores should have enough stock to face that budget sales defined. A minimum of 4 weeks of budget sales is required to be stocked in each store. The actual sales may differ from the budget sales, affecting the decisions of the subsequent weeks.

The sources of costs are the handling of the products in the warehouse and stores, the transportation service, and the holding costs, the latter ones are incurred whenever the capacity limits are exceeded, in the stores and in the warehouse.

2.1. Detailed description

It follows a detailed description of the problem introduced above. The details will then be used to introduce the mathematical notation, which is followed by the mathematical programming model.

Today, the downstream manager makes the decisions based only on levels of stock and sales performance. In this model, in addition to the latter decision variables include the operational costs. The costs include shipping and holding expenses, incurred both in the stores and in the warehouse.

Costs are central to the management of a company. However, for reasons unknown, and not easy to explain, in this company's current practices, the decision making process

[3] We will observe ahead that the data provided doesn't allow this constraint to be respected. The inflow of goods is greater than the outflow.

of the DM does not include or account for costs. The costs that we will introduce here are, usually, only known at the end of the year and, even then, not in much detail. In this work, we aim not only at making the DM's decision process faster but also to allow the consideration of further issues, such as costs. Furthermore, with the tools that are currently used, the manager decision process would be very time consuming, if such issues were included.

The company is a part of a group of companies, logically, the warehouse costs (maintenance and personnel) are split by all companies at the end of the year. The DM has no information of how much is the cost of holding the goods in the warehouse. In addition, it was not possible to obtain that information from the company's managers. This also includes handling costs. The transportation cost was the only value that was provided. Thus, the costs that will be used in this dissertation are the best estimation possible, given the information that was provided. As for the in-store costs, the same problem arises. The stores are, for the most part, located in shopping malls, along with other companies of the group. The storage space is group owned and somehow divided between all companies of the group and then made available to them.

Along the dissertation, given the lack of detailed information and that many costs are out of the scope of the DM's decisions, only include the costs that are incurred as a result of the decisions made by the DM. For example, storage cost is not considered, unless the pre-assigned storage capacity is exceeded. The operations developed by the downstream manager triggers three cost sources:

1. Warehouse – Where the goods are on hold until they are sent to the stores. Here we can identify two types of costs:
    a. Holding costs – These are fixed costs and include the rent, paid to the mother company for the space allocated to the adult fashion division. Regarding the decision making these fixed costs will be ignored as they are incurred regardless the amount of goods store in. However, penalties will be applied whenever the warehouse maximum load is exceeded.

b. Cargo handling costs – To handle the truck loading manpower/machine power is needed. This cost depends on how many boxes are sent, making the DM responsible for it. The warehouse has automated features. This makes the handling time more efficient than that of the stores. Hence, we will assume that it takes 10 minutes (twice as fast as in-store) to handle each box in the warehouse. The latter information, with the income per hour of the warehouse employee gives an estimate of the cargo handling cost.

2. Store – When the goods arrive at the store, they need to be received and properly stored or displayed in the store. It has the same cost types as the warehouse:

    a. Holding costs – The same reasoning as the warehouse is applied, thus they will be ignored. Again, as the DM controls the stock level, a penalty is added to the store costs whenever the storage capacity is exceeded.

    b. Cargo handling costs –The more boxes the manager sends to each store, the more work is required to store the goods. After inquiring store collaborators about how long it takes to receive a box, check the contents, unbox and store, we calculated it to be on average 20 minutes per box. Again, this information together with the hourly salary allow for the calculation of the per box cost.

3. Transportation – delivering the goods from the warehouse to the store has a fixed cost of 10€/per pallet sent. Regardless of the distance from the warehouse to the stores. Each pallet contains on average 10 boxes.

Given this we need to assume that: (i) the holding cost is a sunk cost. The company must pay it no matter how much space is occupied. However, if the maximum capacity is surpassed, a penalty will be applied. The cost, will be the cost per m^2, as published on (*Portaria n.º 156/2014* 2014). Appendix 1 contains the values per zone, as well as the municipalities included in each cluster. The same rationale is applied to the warehouse (2 boxes have one m^2); (ii) each box contains 6 items of each product on average, except

Accessories, which have 12 items per box; (iii) the hourly salary of the store and warehouse employees used will be the minimum hourly pay in Portugal (557€/month ("Decreto-Lei n.º 86-B/2016, de 29 de dezembro," 2016) which is a hourly salary of 3.16€); (iv) due to data constraints, there is only weekly information of sales and stocks, so the algorithm will supply data on a weekly basis; (v) as for the transportation, each truck as a minimum and maximum number of boxes; (vi) the truck limits are the same no matter the destination; (vii) There are lower and upper transportation limits the amount of each product sent to each store, to ensure that there is enough diversity of products in store. Appendix 2 details the latter transportation limits.

2.2. Model formulation

This section introduces the variables and parameters needed to formulate the problem as a mathematical programming model (Table 1), as well as the model.

Mathematical notation	
Indices and Sets	Description
$i \in I$	Product ($I = \{1, \ldots, n\}$).
$k \in K$	Stores, including the warehouse, ($K = \{1, \ldots, m\}$, where m is associated with the warehouse).
t	Time (in weeks) t $\in \{1, \ldots, T\}$.
Variables	
x_{ikt}	Amount (in units) of product i to send to store $k \in K \setminus \{m\}$, in week t.
z_{ikt}	Number of boxes required to send product i to store $k \in K \setminus \{m\}$ in week t. Note that, when referring to the warehouse, i.e., $k=m$ the boxes leave the warehouse and the amount for each product i is given the summation of the boxes of product i sent to all stores ($k \in K \setminus \{m\}$).
s_{ikt}	Stock (in units) of product i available at the beginning of week t in store $k \in K$.

Q_{ikt}	Number of boxes required to stock product *i* at the beginning of week *t* in store *k*.
p_{kt}	Stock over the storage capacity, in boxes, at store *k* in week *t*.
Parameters	
P_{ikt}	Projected (budget) sales in units of product *i* in store $k \in K \setminus \{m\}$ for week *t*.
A_{ikt}	Actual sales of product *i* in store $k \in K \setminus \{m\}$ in week *t*, this value is only known in week *t*+1.
US_k	Total storage capacity, in boxes, of store *k*.
OO_{it}	On order quantity, in boxes, of product *i* in week *t*. Amount of goods that were bought and arrive at the warehouse in week *t*.
CH_k	Handling cost per box in store *k*.
CR_k	Rent cost per m^2 in store *k*.
TC	Transportation cost per box.
UM	Maximum transport load, in boxes, per week (all products) to all stores.
UN_{ik}	Maximum transport load, in units, of product *i* to store $k \in K \setminus \{m\}$.
LN_{ik}	Minimum transport load, in units, of product *i* to store $k \in K \setminus \{m\}$.
C_i	Units per box of product *i*.

Table 1 - Notation

Recall that the model presented below is a weekly model, therefore the inventory held at the stores and warehouse at the beginning of the week is given as any other input, while the one for the following week is an output to be used when the week t+1 decisions are addressed. Therefore, at the beginning of each week the inventory value will be updated for the stores as $\mathbf{s_{ikt}} = \mathbf{s_{ikt}} + \mathbf{P_{ik(t-1)}} - \mathbf{A_{ik(t-1)}}$, since when it was first computed only the predicted sales were known. Regarding the warehouse, its true value will be higher due to the on order quantity, therefore it will be updated as $\mathbf{s_{imt}} = \mathbf{s_{imt}} + \mathbf{OO_{it}}$. Only variables x_{ikt} are decision variables, since all other variables are determined by their value.

However, these auxiliary variables are helpful to write the mathematical programming model, making it easier to write and understand.

Using the notation and assumptions previously introduced, we now are able to present the mathematical formulation of the problem. Equation (1) describes the objective function.

$$Minimize \quad C_t = TC \sum_{i=1}^{n} \sum_{k=1}^{m-1} z_{ikt} + \sum_{k=1}^{m} CH_k \sum_{i=1}^{n} z_{ikt} + \sum_{k=1}^{m} \frac{CR_k \times p_{kt}}{2} \tag{1}$$

Subject to:

$$z_{ikt} \geq \frac{x_{ikt}}{\mathrm{C_i}}, \qquad \forall i \in I, k \in K \backslash \{m\}, \tag{2}$$

$$z_{imt} = \sum_{\mathrm{k=1}}^{\mathrm{m-1}} \mathrm{z_{ikt}}, \qquad \forall i \in I, \tag{3}$$

$$Q_{ikt} \geq \frac{s_{ikt} + x_{ikt}}{\mathrm{C_i}}, \qquad \forall i \in I, k \in K \backslash \{m\}, \tag{4}$$

$$Q_{imt} \geq \frac{s_{imt} - \sum_{\mathrm{k=1}}^{\mathrm{m-1}} x_{ikt}}{\mathrm{C_i}}, \qquad \forall i \in I, \tag{5}$$

$$p_{kt} \geq \sum_{\mathrm{i=1}}^{\mathrm{n}} \mathrm{Q_{ikt}} - US_k, \qquad \forall k \in K \backslash \{m\}, \tag{6}$$

$$p_{mt} \geq \sum_{\mathrm{i=1}}^{\mathrm{n}} (Q_{\mathrm{imt}} + OO_{it}) - US_m, \tag{7}$$

$$\mathrm{s_{ikt}} + x_{ikt} \geq \sum_{\mathrm{u=t}}^{\mathrm{t+3}} \mathrm{P_{iku}}, \qquad \forall \mathrm{i} \in \mathrm{I}, \mathrm{k} \in \mathrm{K}, \tag{8}$$

$$p_{kt} \leq 0.1\, US_k \qquad \forall k \in K, \tag{9}$$

$$x_{ikt} \leq UN_{ik} \qquad \forall i \in I, k \in K \backslash \{m\}, \tag{10}$$

$$x_{ikt} \geq LN_{ik} \qquad \forall i \in I, k \in K \backslash \{m\}, \tag{11}$$

$$\sum_{i=1}^{n} x_{ikt} \leq UM, \qquad \forall k \in K \backslash \{m\}, \tag{12}$$

$$\mathrm{s_{ik(t+1)}} = \mathrm{s_{ikt}} + x_{ikt} - \mathrm{P_{ikt}} \qquad \forall i \in I, k \in K \backslash \{m\}, \tag{13}$$

$$\mathrm{s_{im(t+1)}} = \mathrm{s_{imt}} - \sum_{\mathrm{k=1}}^{\mathrm{m-1}} x_{ikt}, \qquad \forall \mathrm{i} \in \mathrm{I}, \tag{14}$$

$$x_{ikt}, z_{ikt}, Q_{ikt} \geq 0 \text{ and integer} \qquad \forall i \in I, k \in K, \tag{15}$$

$$s_{ikt}, p_{kt} \geq 0 \qquad \forall i \in I, k \in K. \tag{16}$$

Equation 2 - Objective function and constraints.

Constraints (2) to (7) enforce the auxiliary variables definition. Inequalities (2) together with the fact that the *z* variables are integer, force them to be at least the smallest number of boxes required to fit the amount of each product sent to each store. Equalities

(3) calculate the number of boxes per product to leave the warehouse as the total number of boxes, per product, that is sent to the stores. Similarly, inequalities (4) and (5) determine the number of boxes that the stores and the warehouse, respectively, have to hold as inventory. Finally, the number of boxes in inventory that are over the capacity limits, if any, is determined by inequalities (6) and (7) for the stores and for the warehouse, respectively.

Constraints (8) force each store to have enough of each product to satisfy the predicted demand for four weeks; however, the total inventory of each store cannot be more than 10% over the storage capacity. The latter constraints are given by inequalities (9).

The number of boxes that each store receives has upper and lower limits both regarding each product, inequalities (10) and (11). The total upper limit of units sent, inequality (12).

Equations (14) and (15) are the balance equations for the stores and warehouse, respectively.

Finally, the nature of the variables is given by constraints (16) and (17). Note that, for the variables associated with stock (*s* and *p* variables) it is enough to define them as non-negative. Since they are obtained by adding and subtracting integer values, they will always have integer values.

3. Solution approach

To solve the stock distribution problem, we resorted to a genetic algorithm. What we propose is an algorithm built from scratch adapted to the problem previously described. Although, with some modifications, it may be used on other similar problems.

The complexity and difficulty of the problem requires the use of a heuristic in order to find solutions within a reasonable computational time. The early works on heuristics to solve complex lot sizing problems are from Silver and Meal (1973). Their heuristic aims at determining the production requirements at a minimum cost, considering setup and holding costs. An average cost is calculated and the computation stops whenever the current period cost is higher than the last period (i.e., C(T) > C(T-1), where C is the average cost and T is the period) – the Silver Meal criterion (Jans & Degraeve, 2004). An extension of the previous heuristic is the Least Unit Cost (LUC). Instead of evaluating the cost at a given period it considers the cost per unit. The order size increases until the per unit production cost increases (Wee & Shum, 1999).

Karimi et al. (2003), after reviewing various algorithms suggested by multiple authors, advise that using heuristics to solve hard lot sizing problems greatly benefits the researchers. The authors call for an increase of the study of complex and realistic problems that will ultimately require the use of meta-heuristics and further extend the reach in the literature of such approaches.

Several authors developed genetic algorithms to solve lot-sizing problems, mainly in an industrial context (Sikora, 1996; Lee, Sikora, & Shaw, 1997; Kimms, 1999; Xie & Dong, 2002). The results show that GA is able to solve complex problems efficiently and that it outperforms other heuristics, such as tabu search (Kimms, 1999). The combination of multiple heuristics (hybrid heuristics) can also improve the solutions and computational efficiency, for example Lee et al. (1997) improved their GA by adding a simulated annealing (SA) process to control convergence.

A genetic algorithm is a metaheuristic based on natural selection ideas and was first introduced by Holland (1975). In a GA, a population of solutions to a problem evolves towards a better solution. Each individual of the population (which is a solution and also called phenotype) has a set of properties that define it (chromosome or genotype). The

evolution of the population occurs iteratively through genetic operations that will be explained ahead.

Let us take the example of rabbits as an illustration of the process summarized above (Khouja, Michalewicz, & Wilmot, 1998; Sarker & Newton, 2002). In a given population of rabbits, some are faster and/or smarter than others. The later ones are less likely to be eaten by predators, consequently being more likely to reproduce. However, some dumb and slow rabbits also survive, even if only due to luck. The breeding process results in a mixture of multiple rabbit genetic material. Some slow rabbits breed with faster ones, some smart with dumb ones and so on. The resulting generation is, on average, smarter and faster. Plus, nature may throw a wild card that mutates the rabbit's genetic material. While explaining how we designed the algorithm, we will describe some features that we tested. However, for further information on GAs and a more comprehensive description of the heuristic see "Genetic Algorithms in Search, Optimization and Machine Learning" by David E Goldberg (2006). The book provides an overlook on GAs, as well as more advanced concepts and application of the algorithm across different fields.

Normally a genetic algorithm performs the following steps:

1. Initialize the population with a certain number of solutions. Although there is no strict rule when it comes to size, the bigger the population the higher the diversity of solutions available.
2. Calculate the "quality" of the individuals using a fitness function.
3. Select some solutions and apply genetic operators to them. A probability of crossover and/or mutation may be set for each pair of selected solutions. Generally, the crossover operation has a 100% probability of occurring, while the mutation occurs with a low probability, e.g, 1/(string length of the individual) (Sarker & Newton, 2002).
4. Replace the population with the new population.
5. Repeat steps 2-4 until the stop criteria is met – it can either be a good enough solution or a maximum number of iterations/time

To describe how the algorithm was built we will break this chapter into five sections that combined, will explain how the tool (algorithm) used to solve the problem operates:

- Problem representation
- Initial population of solutions
- Evaluations of the individuals (fitness function)
- Genetic operators
- Values of the parameters

3.1. Problem representation

The problem is represented by an array that contains a possible combination of quantities, of each product, for each store. The genes are the quantities of each product to be sent. There are as many genes as products. For example, as we can see in Figure 2, the first individual (solution) suggests not sending Product 1 to Stores 1 and 2, yet to Stores 3 and 4 proposes shipping 72 and 147 units, respectively. Each row of the array represents a store and thus there are as many rows as stores to be supplied, four in the example being used (see Figure 2). The combination of the quantities sent to all stores is the solution to our problem. The columns, that we will call individuals, are the possible solutions. In the given example, we have three individuals that give us the quantities to send of each of the eight products to each of the four stores.

```
[
[[0, 33, 0, 530, 0, 0, 3, 0]    [0, 33, 0, 530, 219, 256, 0, 0]    [0, 132, 0, 32, 0, 76, 97, 67]]
[[0, 68, 77, 139, 83, 434, 37, 25]    [0, 68, 77, 139, 83, 434, 37, 25]    [0, 68, 77, 139, 83, 434, 37, 25]]
[[72, 56, 115, 225, 113, 429, 126, 15]    [72, 56, 115, 225, 113, 429, 126, 15]    [72, 56, 115, 225, 113, 429, 126, 15]]
[[147, 0, 28, 235, 241, 105, 8, 34]    [147, 0, 28, 235, 241, 105, 8, 34]    [147, 0, 28, 235, 241, 105, 8, 34]]
                                                                                                                    ]
```

Figure 2 - Problem representation example.

The choice of representation was the first step in constructing the algorithm. Moreover, the level of knowledge on the programming language was also in its initial steps. Hence

the choice of value encoding[4]. Besides easing the interpretation, the chromosomes decodes themselves.

Nevertheless, we have explored two other representation methods. The first involved using floating point numbers between 0 and 1 instead of integers. To convert the genes into quantities a decoder was necessary. The final quantity would be the weighted average of each product multiplied by some criteria, the truck capacity for example, but only genes above a certain threshold would be considered. For example, consider the chromosome depicted in Figure 3 and assuming a threshold of 0.3, products 1, 3, 7 and 8 would not be sent. For the remaining products, the quantity to send would be related to their weight in the chromosome. However, this approach required a decoder and encoder function every generation.

[0.2521, 0.7223, 0.1243, 0.8754, 0.9219, 0.5256, 0.0135, 0.1135]

Figure 3 - Floating point chromosome example.

The other option to address the issue of representation was a priority based system. Gen, Altiparmak, and Lin (2006) applied such representation structure on a single product transportation problem. In short, given the transportation capacity, demand and costs, the clients that have higher priority (lowest costs) are the ones whose demand is satisfied first. We propose this as a future improvement of the algorithm, although not as a representation, but rather as a repair mechanism. In case of shortage of available stock, in the warehouse, the algorithm must have a way to choose how to distribute the products.

3.2. Initial population

As a search problem, to better explore the solution space, the GA needs its population spread out through it. So, the first step of a GA is to generate a population big enough to cover the solution space. Therefore, we generate the initial population randomly, not only to ensure the exploration of the solution space, but also to avoid an early convergence (Leung, Gao, & Xu, 1997). A good initial population can improve the performance of a GA (Zitzler, Deb, & Thiele, 2000; Burke, Gustafson, & Kendall, 2004; Lobo & Lima,

[4] Value encoding is a type of solution representation. The chromosomes are represented by strings of values that are connected to the problem. Other types of encoding include, the most commonly used, binary (string of bits) and permutation (string of numbers that represent a sequence).

2005), thus, we seeded information that ensures that the individuals achieve good fitness scores in the early stages (Rajan & Shende, 1999; Casella & Potter, 2005).

Figure 4 illustrates how we generate the initial population. We need the population parameters, i.e., number of stores, number of individuals and size of each chromosome (number of products). As previously stated, to ensure good initial fitness scores, we set the initial parameters of the population as the transport bounds. These limits don't allow the random generator to produce big numbers that would compromise the individuals at the start of the execution. In the algorithm, the chromosomes are composed of two parts. The first one is binary, and represents the decision of whether to send or not a product. Despite its importance to the operation of the algorithm, this part is excluded from the final representation. We removed this part, because all the 1's correspond to all the genes that on the chosen representation are greater than zero, while the 0's are all the genes with null quantities. After the generation of the two components, the initial population is the result of the concatenation of both parts.

procedure 1: Initialize population
inputs: N_POP : 1st level population size (no. of stores)
POP_SIZE : 2nd level population size (no. of possible solutions for each store)
CHROMO_SIZE : no. of genes in each chromosome
LM_k : lower bound of the quantity of product *k* to be transported
UM_k : upper bound of the quantity of product *k* to be transported
output: population: random population of N_POP × POP_SIZE chromosomes with size CHROMO_SIZE each
step 1. z ← random **binary** array of size N_POP × POP_SIZE chromosomes. The chromosomes have a CHROMO_SIZE/2 length
step 2. y ← random **integer** between, LM_k and UM_k, array of size N_POP × POP_SIZE chromosomes. The chromosomes have a CHROMO_SIZE/2 length.
step 3. population ← concatenate z and y

Figure 4 – Pseudocode: initial population.

3.3. Fitness function

The fitness function is an important backbone of the GA, if not the most important. The fitness function is responsible for evaluating how good the individuals are to answer the problem. Traditionally, low fitness is good in minimization problems and the opposite in maximization problems. The fittest solutions have higher probability (not guaranteed) of being passed on to the next generation. Hence the pivotal part played by the fitness function on the performance of the genetic algorithm. Ultimately, the best possible choice

(quantitative and qualitative) will arise from experience or trial run optimization (Haupt & Haupt, 2004).

The choice and implementation of the fitness function is a difficult step when constructing a GA. Different functions have different impacts in the GA (Grefenstette & Baker, 1989). Since the fitness function has no conventional format we chose to adopt a different approach. Being a cost minimization problem, the fittest individuals should be those that achieve the lowest costs. However, the fittest individuals may not be admissible, given the problem constraints. We opted to evaluate the individuals not only based on their cost, but also on their compliance with the problem constrains. This way we are able to classify the individual's admissibility and alignment with the objective.

To evaluate the individuals, we use a "score system". The score is awarded if the individual can meet a constraint. The individual is awarded 10 points per constraint that is satisfied. However, there are restrictions that can have a slack of no more than 10% of the value of the restriction. If the individuals fall within the 10% limit, 5 points are awarded. If the individual is in the 50% lowest cost individuals, is awarded 5.400 points (108 stores × 50 points). Table 1 summarizes the score system. Each of the point represented below are the point awarded to each chromosome, since many restrictions are evaluated at the chromosome level (i.e. store level).

Score System	
Points	Description
50 points	Only awarded to solutions that are on the 50% lowest costs.
10 points	Awarded to solutions that comply with a restriction.
5 points	Intermediate points awarded if a solution is 10% below/above a restriction. Only applied in restrictions which the excess/shortage are not highly punitive/costly.

Table 2 - Score system details.

To further understand why and how we apply the framework, we will now describe the restrictions used. The latter ones seek to guarantee that the best solutions are also admissible.

Constraint 5 tackles the issue of the availability of product in the warehouse (see Figure 5). The current stock in the warehouse must be enough to cover the quantities sent. This case has no intermediate point system. For each product either the sum the amounts sent to all stores in the warehouse, and the 10 points are awarded, or it is not, and no points are given.

procedure 2.1: Fitness function (constraint 5)
inputs: population :population to be evaluated
s_{imt} :current stock of each product in the warehouse
$\sum_k x_{ikt}$:quantity to be sent to all stores of all products
output: score: score relative to the compliance of constraint 5
step 1. $\sum_k x_{ikt}$ ← sum of the total quantities to be sent of all stores, per product
step 2. Compare $\sum_k x_{ikt}$ with the current stock on the warehouse (s_{imt}). If the former is lower than the latter, award 10 points to all stores, on the individual being evaluated. Otherwise no points are given

Figure 5 – Pseudocode: awarding points regarding constraint (5).

Constraint 8 ensure that the stock in-store is enough to cover four weeks of (projected) sales. We need the sales budget for the forthcoming four weeks (P_{ikt}, $P_{ik(t+1)}$, $P_{ik(t+2)}$, $P_{ik(t+3)}$), the current stock of each store (s_{ikt}) and the quantity to be sent (x_{ikt}). For this constraint 10 points are awarded to each store whenever the projected stock is enough to cover the aforementioned four weeks. On the other hand, if the stock is at least 90% of the needed quantity, only 5 points are awarded and 0 otherwise (see Figure 6).

procedure 2.2: Fitness function (constraint 8)
inputs: population :population to be evaluated
$\sum_{u=t}^{t+3} P_{iku}$:budget sales for week t (current), t+1, t+2 and t+3
s_{ikt} :current stock, per store, per product
x_{ikt} :quantity to be sent, per store, per product
output: score: score relative to the compliance of constraint 8
step 1. rest_1 ← list containing the sum of the quantities of the four periods in respect to each store and product
step 2. compare rest_1 with the current stock plus the quantity to be sent ($s_{ikt} + x_{ikt}$)
step 3. if ($s_{ikt} + x_{ikt}$) ≥ rest_1, add 10 points to the fitness of the option. However if ($s_{ikt} + x_{ikt}$) ≥ 0.9 × rest_1 add 5 points. Otherwise no points are awarded

Figure 6 – Pseudocode: awarding points regarding constraint (8).

Constraint 9 impose a limit of the stock held. The 10 points are awarded if the storage capacity is not exceeded, however 5 points are added if the limit is surpassed by, at most, 10%. The latter, although not ideal, is possible and can be dissipated within a day or two of sales. If the 10% slack is exceeded, no points are awarded (see Figure 7).

procedure 2.3: Fitness function (constraint 9)
inputs: population :population to be evaluated
US_k :storage capacity of each store
p_{kt} :stock over capacity, per store
output: score: score relative to the compliance of constraint 9
step 1. if $p_{kt} = 0$ add 10 points to the fitness. If $p_{kt} \leq 0.1 \times US_k$ add 5 points. Otherwise no point is awarded

Figure 7 – Pseudocode: awarding points regarding constraint (9).

Constraints 10 and 11 limit the minimum and maximum amount, per product, per store. The quantity to send to each store must be within these limits. Like the previous constraints, these ones cannot be split. The chromosome gets 20 points if it falls between both limits. Otherwise no point is awarded (see Figure 8).

procedure 2.4: Fitness function (constraints 10 and 11)
inputs: population :population to be evaluated
x_{ikt} :quantity to be sent, per product, per store
LN_{ik} :minimum to send, per product, per store
UN_{ik} :maximum to send, per product, per store
output: score: score relative to the compliance of constraints 10 and 11
step 1. if $LN_{ik} \leq x_{ikt} \leq UN_{ik}$ add 20 points to the chromosome's score. Otherwise nothing is added

Figure 8 – Pseudocode: awarding points regarding constraints (10) and (11).

Constraint 12 is the total transport upper limits. Since a limited number of trucks is available to transport the products to the store, no intermediate points are given. If the limit is respected 10 points are added, otherwise no point is added (see Figure 9).

procedure 2.5: Fitness function (constraint 12)
inputs: population :population to be evaluated
$\sum_i \sum_k x_{ikt}$:quantity to be sent of all products, to all stores
UM :maximum quantity to send
output: score: score relative to the compliance of constraint 12.
step 1. if $\sum_i \sum_k x_{ikt} \leq$. UM add 10 points. Otherwise none are given

Figure 9 – Pseudocode: awarding points regarding constraint (12).

The last condition that we award points is the cost of the individual. The 50% lowest cost individuals get 5.400 point each (i.e., 50 points × 108 stores). The cost of each individual is calculated prior to the fitness function.

procedure 2.6: Fitness function (cost score)
inputs: population :population to be evaluated
cost :array with the cost of each individual of the population
output: score: score relative to the cost
step 1. z ← POP_SIZE (no. of individduals) × 50%
step 2. From cost get the lowest z costs and add 50 points to each store in the individual

Figure 10 – Pseudocode: awarding points regarding the cost score.

After all constraints are evaluated the fitness score is the sum of all these scores. Note that many scores are given for every gene that respects a constraint. For example, if every gene, of every chromosome in an individual meets restrictions 10 and 11, it would add 17.280 points (20 points × 108 stores × 8 products) to the score of the individual.

3.4. Genetic operators

Picking up the rabbit's example given initially we have to evolve the population through the breeding process. Thus we need to apply genetic operators to combine existing solutions into others or to generate diversity (Merelo & Prieto, 1996). There are three genetic operators that are used in every GA: Selection, Crossover and Mutation.

Apart from the mutation operator, we apply standard genetic operators. The population that will make up the subsequent generation will be composed of a group of elites (individuals that have better fitness scores), the offspring that result from the crossover, and a new group of random individuals (mutants) – See Figure 11.

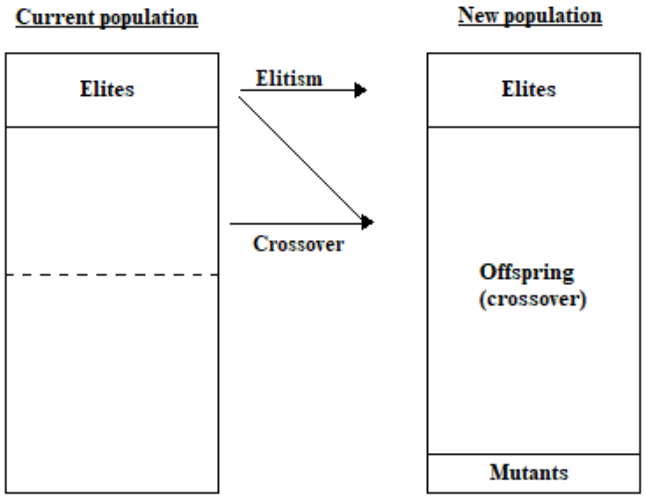


Figure 11 - New population example.

3.4.1. Selection

Selection is used to select the fittest individuals for the next generation based on a fitness function and/or probability (Khouja et al., 1998). In addition, this function is also used to select the parents to be used by the crossover operator.

The literature hasn't agreed on what is the best parent selection method, depending on the problem, tournament selection and roulette wheel are chosen over the random choice. Regarding lot sizing problems, different authors use different methods. Kimms (1999) creates a pool with the solutions that have better fitness (elites) and then chooses the parents randomly out of the elite pool. However, roulette wheel selection seems to be the most commonly used (Gaafar, 2006; Sarker & Newton, 2002; Xie & Dong, 2002). We applied a similar approach to Kimms. The chromosomes that have above average scores are placed in a pool. From the pool of best performing chromosomes, we pick the parents randomly.

The elite consist of the individuals of a population that have the best fitness scores. Elitism ensures that the best solutions are not lost during the disruptive operation of crossover or mutation. By preserving the best individuals of each generation we can speed up the performance of the GA (Rudolph, 2001; Zitzler et al., 2000). The elitism process is simple. We pick a percentage of the individuals with the highest sum of fitness scores.

The individuals are then copied onto the next generation (Altiparmak, Gen, Lin, & Paksoy, 2006; Dellaert, Jeunet, & Jonard, 2000; Khouja et al., 1998).

In terms of execution of the process, the elite function of the algorithm proposed in this dissertation does not copy directly the result into the next generation. It creates an elite group that will be later merged with the crossover and mutation groups. The result will then be the subsequent generation. The pseudo code of the elitism is given in Figure 12.

procedure 3: Elite population
inputs: population w/ scores :population with the scores assigned
%_elites :percentage of the population to be considered elite
output: elite population: the elite population that will be copied into the next generation
step 1. %_elites ← Get the percentage of individuals to consider elite
step 2. n_elite ← Get the number of solutions to copy to the next generation (%_elite × POP_SIZE)
step 3. scores ← Get the score of each individual
step 4. ind_max ← Search in scores for n_elite solutions with the highest scores
step 5. elites ← Copy the ind_max solutions

Figure 12 – Pseudocode: Elite function.

3.4.2. Crossover

Crossover is applied after two parents are selected, some chromosomes are exchanged to generate the offspring (Melanie, 1999). The crossover technique allows the convergence towards a solution, further exploring the subspace (Grefenstette, 1986).

One of the most popular crossover technique is the 2-point crossover. Jong (1975) and David E. Goldberg (1989) favor the 2-point crossover over the 1-point crossover and multi-point crossover. The n-point crossover selects n points on the chromosome and the information between those points are exchanged between the parents. Figure 13[5] illustrates the use of 1-point (left) and 2-point (right) crossovers.

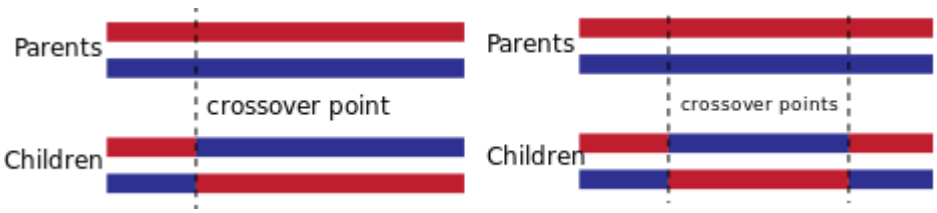


Figure 13 - 1-point crossover (left) and 2-point crossover (right) examples.

[5] Credits of the figure: R0oland (https://commons.wikimedia.org/wiki/File:OnePointCrossover.svg and https://commons.wikimedia.org/wiki/File:TwoPointCrossover.svg)

As we said previously, operationally, the algorithm contains a binary component, which represents the decision to send. When we applied the 2-point crossover on the whole chromosome two problems arose. On one half of the chromosome the decision was to send, but no quantities were associated to that decision. The problem is illustrated in Figure 14, marked with the letter A. The other problem was having a quantity to send, but the decision was null (Figure 14, letter B).

To overcome these issues, we need to see both halves as two dependent chromosomes. Thus, the cuts that we make on one, have to be mirrored on the other. Given the size of both halves, and to avoid disrupting the building block[6] of the chromosome, we used the 1-point crossover.

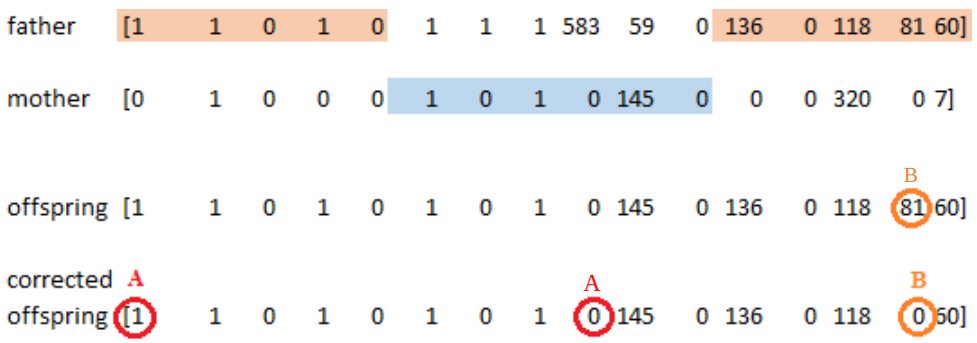


Figure 14 - 2-point crossover issue.

We already introduced how the parents are chosen. Furthermore, a random generator will decide which parent gets the first half of the crossover. If the number is lower than 0.5 the "random father" will provide the first half and the "random mother" will fill in the rest. Otherwise, the "mother" gets the first half and the "father" the rest. The parent selection and crossover pseudocode is given by Figure 15.

[6] Building blocks are short, low order and high performing schematas (chromosome's masks) (Beasley, Bull, & Martin, 1993; David E. Goldberg, 1989).

procedure 4: Crossover
inputs: population w/ scores :population with the scores assigned
%_crossover :percentage of the new population the be the result of crossover
output: offspring: the offspring resultant of the crossing of two parents from the population. These will be part of the new generation.
step 1. %_crossover, n_crossover← Get the percentage of individuals that will be on the next generation resulting from crossover and respective number.
step 2. parent_pool ← Gather the chromosomes that have scores above average.
step 3. rnd_father, rnd_mother ← Get a pair of random parents from the parent_pool.
step 4. If rnd_father = rnd_mother, repeat step 3.
step 5. rnd_factor ← Get a random float-point number between 0 and 1.
step 6. If the random factor is < 0.5 the rnd_father chromosome will be on the 1st and 3rd quarters of the child's chromosome, and the rnd_mother the remaining. Otherwise, rnd_mother chromosome will be on the 1st and 3rd quarters of the child's chromosome, and the rnd_father the remaining.
step 7. child ← add the offspring to the child array. The array will then be merged with the elite and mutants to form the new population

Figure 15 – Pseudocode: selection of parents and crossover.

3.4.3. Mutants

We use the mutants as an alternative to mutation. Mutation randomly changes (flips) the chromosome (Haupt, 1995). It avoids early convergence to a solution, allowing a broader exploration of the solution space. Instead of changing each chromosome we introduce new random individuals in the population – the mutants or immigrants.

This technique was first introduced by Bean (1994) and used by Toso and Resende (2015) on their biased random-key genetic algorithm. Traditional mutation, when used, disrupts the results of the crossover, that is why it is used with very low probabilities (< 1%). Thus, we are giving a greater role to this operator, and not sacrificing the results of the other genetic operators. These new individuals are created using the same procedure used in generating the initial population. The pseudocode to the mutant procedure is shown by Figure 16.

procedure 5: Mutation
inputs: %_mutants :percentage of the new population to be mutants
N_POP :1st level population size (nº of stores)
CHROMO_SIZE :nº of genes in each chromosome
output: mutants: the mutants to be inserted in the new generation
step 1. %_mutants, n_mutants ← Get the percentage and number of individuals, respectively, that will be in the next generation as mutants.
step 2. w ← random **binary** array of size N_POP × n_mutants chromosomes. The chromosomes
have a CHROMO_SIZE/2 length
step 3. v ← random **integer** array of size N_POP × n_mutants chromosomes. The chromosomes have a CHROMO_SIZE/2 length
step 4. mutants ← concatenate w and v

Figure 16 – Pseudocode: Mutation..

3.4.4. Parameters

To set the parameters we tested values provided by several authors. However, considering we used a different approach with the mutation operator, we will source the parameters of the latter ones from different authors than the population size and generation parameters. The tests were carried out on a machine running Windows 10 64 bits on 2 2.2GHz Intel® Core™ i5-5200U CPUs with access to 6 Gb RAM. The algorithm runs in Python 3.6.0.

The combinations of the parameters of the genetic operators used are shown in Table 2. Both Toso & Resend and Bean used the mutants successfully in their work so we decided to test the values they have used. However, a third combination is introduced that maintains Toso and Resende's elite parameter, while giving more weight to the crossover.

Genetic Operators Parameters				
Combination no.	Crossover	Elite	Mutants	Author
1	0.75	0.15	0.10	(Toso & Resende, 2015)
2	0.79	0.20	0.01	(Bean, 1994)
3	0.80	0.15	0.05	-

Table 3 - Combinations of the genetic operators' parameters.

In addition to the genetic operators' parameters, three different population sizes and three generations numbers were tested. The optimal population size is a hard problem

(Eiben, Hinterding, & Michalewicz, 1999), as many factors need to be taken into account, such as: problem difficulty, number and diversity of individuals, and search space among others (Diaz-Gomez & Hougen, 2007). The population size varies between 30 (Xie & Dong, 2002) up to 200 (Khouja et al., 1998). But as we will see, the higher the number of individuals, the slower is the algorithm processing each generation. Table 4 shows the three values of population sizes tested.

Population Size Parameter		
Combination no.	Population Size	Author
1	30	(Xie & Dong, 2002)
2	40	-
3	60	-

Table 4 - Population parameter values.

The number of generations is also an issue that is highly dependent on the problem. Deb and Agrawal (1998) in their work on the interactions among GA parameters, show that large populations require a low number of generations to return good solutions, and vice versa.

Generations Parameter		
Combination no.	Generations	Author
1	200	(Sarker & Newton, 2002)
2	800	-
3	2,000	-

Table 5 - Generations Parameter.

This setup results in 27 different combinations of parameters. The tests will be done under the same initial setup and run over 25 weeks. The four combinations that yield the lowest costs will run over the 52 weeks and be compared with the method currently being applied by the company. Appendix 3 summarizes the 27 possible combinations.

The tests that achieved the lowest costs were tests 2, 4, 8 and 10 (see Table 6). The results show that the algorithm is able to converge rapidly (3 out of the 4 best tests run only 200 generations). Moreover, the best genetic operator combination is split between Bean and Toso and Resende. Appendix 4[7] summarizes the results in a table and in Appendix 5 the results can be visualized.

Test results					
Test no.	Generations	Population	Crossover	Elite	Mutants
2	200	30	0.79	0.20	0.01
4	200	40	0.75	0.15	0.10
8	200	60	0.79	0.20	0.01
10	800	30	0.75	0.15	0.10

Table 6 - Top four test results.

Every test starts with about the same costs. However, by week 13, most of the tests start to increase their costs. These surges are caused by excess of stock in the stores, and later, in the warehouse. The best performing tests seem to send slightly more products over the first weeks (slightly higher costs). Hence, it avoids having large amounts of stock in the warehouse to when large on order quantities arrive.

Figure 17 compares tests 4 and 10 (top performing tests) against tests 5 and 12 (underperforming tests). The surge of the latter ones is not in week 13, as mentioned above, but in week 17 and 18, respectively.

[7] The data of some tests is missing because the increasing costs didn't justify extending the time executing the test.

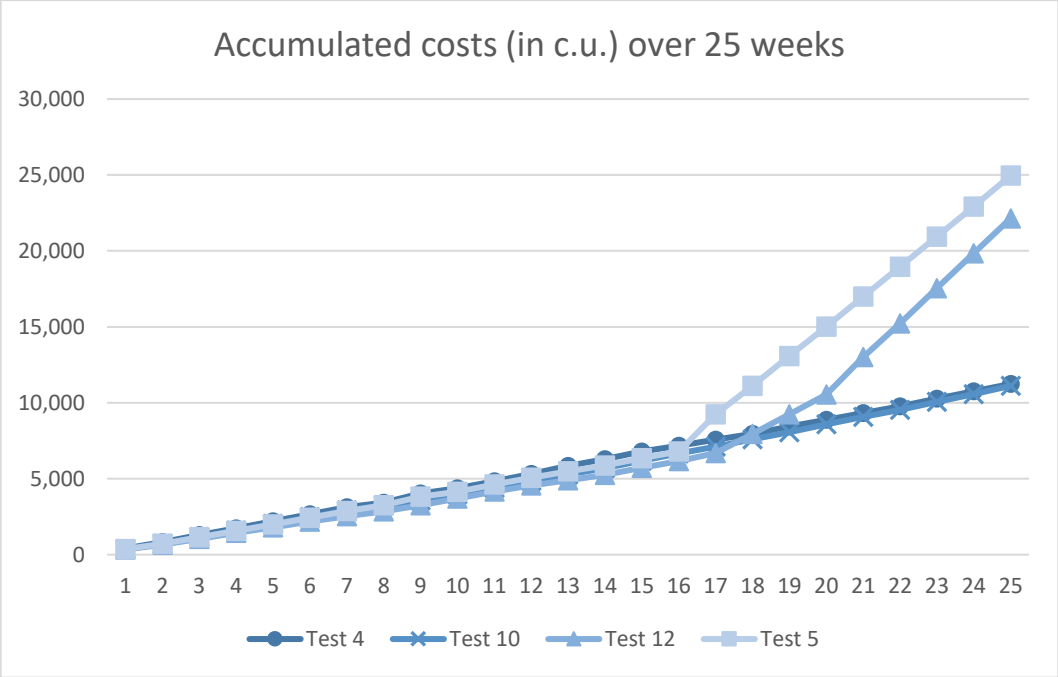


Figure 17- Comparison of two good performing tests with two underperforming tests.

To select the parameters, we carried out 27 different tests. The algorithm run on 25 different instances of time, i.e. for 25 weeks. In Chapter 4, the best 4 combinations of parameters were then tested during the 52-weeks period and compared with the results of the system that is currently used. We will, henceforth, refer to the tests as combinations.

3.5. Repair function

A repair function is used on the top five solutions, after all the generations are processed, i.e., at the end of each week. The function aims at amending misalignments with the problem restrictions. Although the fitness function also evaluates the admissibility of the solutions, we observed that as the problem progresses in the time, some limitations are, inevitably, not being met (i.e., capacity constraints). Thus, to provide the best solution possible a repair mechanism was introduced. Conditions are addressed in order of importance and once verified, and corrected if needed, they cannot be changed. That is, conditions, verifications and corrections cannot override the previous conditions, and cannot be overridden by the subsequent conditions.

We set as first priority the warehouse capacity. If the capacity is exceeded (recall that for this condition, the predicted on order quantity is accounted for), a coefficient greater

than 1 is applied to all the stores quantities. The reason behind this choice is that, as we will see in the next chapter, sending the stock to the store is more efficient, in terms of costs. The pseudocode used to verify the warehouse inventory and eventually correct it is given in Figure 18.

procedure 6.1: Repair function (warehouse capacity)
inputs: population : population to be repaired
US_k :storage capacity of the warehouse
$\sum_i s_{ikt}$:current stock of all products, on the warehouse
x_{ikt} :quantity to be sent per product, per store
p_{mt} : number of boxes in inventory that are over the warehouse capacity limits
UM_{ik} :maximum transport load, per product, per store
output: verify and amend the warehouse capacity constraint
step 1. coef $\leftarrow \sum_i s_{iMt} / US_k$ get the coefficient that will be used to send enough quantity to avoid exceeding the warehouse capacity
step 2. If $p_{mt} > 0$ execute step 3, otherwise do nothing
step 3. If $x_{ikt} > 0$:
$x_{ikt} \leftarrow x_{ikt} \times$ coef
Otherwise:
$x_{ikt} \leftarrow UM_{ik}$

Figure 18 – Pseudocode: warehouse inventory repair function.

The inventory capacity of the stores is then checked and quantities sent are rectified whenever needed (see Figure 19).

procedure 6.2: Repair function (store capacity)
inputs: population : population to be repaired
$\sum_i x_{ikt}$:quantity to be sent of all products, per store
output: verify and amend the store capacity constraint
step 1. If $p_{kt} > 0$ and $p_{mt} = 0$ then $\sum_i x_{ikt} \leftarrow 0$, otherwise do nothing

Figure 19 – Pseudocode: store inventory repair function.

Additionally, to ensure that the stores do not run out of stock, we verify if the stock available is enough for four weeks of budgeted sales. In case this condition is not met, the quantity to be sent needs to be increased. This is done by adding the difference between the four week budget sales and the current stock in the store (see Figure 20). Recall that this verification and eventual correction will only execute if the previous conditions did not execute.

procedure 6.3: Repair function (condition 3)
inputs: population :population to be repaired
$\sum_{u=t}^{t+3} P_{iku}$:budget sales for week t (current), t+1, t+2 and t+3
s_{ikt} :current stock, per store, per product
x_{ikt} :quantity to be sent, per store, per product
output: verify and amend the budget constraint
step 1. budget ← list containing the sum of the quantities of the four periods in respect to each store and product
step 2. If $p_{kt} = 0$ and $p_{mt} = 0$ execute step 3. Otherwise do nothing
step 3. If $\sum_{u=t}^{t+3} P_{iku} - s_{ikt} < US_m$. execute step 4. Otherwise do nothing
step 4. If $(s_{ikt} + x_{ikt}) < \sum_{u=t}^{t+3} P_{iku}$ then $x_{ikt} \leftarrow \sum_{u=t}^{t+3} P_{iku} - s_{ikt}$, otherwise do nothing

Figure 20 – Pseudocode:4 week sales repair function.

Lastly, it is ensured that the transport limits are met. If the value is above the upper limit, then some or all quantities to be sent need to be decreased while if it is below the lower limit the reverse is true. This condition executes independent of the others, i.e., it can override previous corrections. Figure 21 contains the pseudocode to the fourth condition being evaluated by the repair function.

procedure 6.4: Repair function (condition 4)
inputs: population :population to be evaluated
x_{ikt} :quantity to be sent, per product, per store
LM_{ik} :minimum to send, per product, per store
UM_{ik} :maximum to send, per product, per store
output: verify and amend the transport limits constraint
step 1. to_send ← Get the quantities per product, per store
step 2. LM_{ik}, UM_{ik} ← Get the minimum and maximum transport capacities
step 3. $x_{ikt} < LM_{ik}$. If the condition is true $x_{ikt} \leftarrow LM_{ik}$, otherwise do nothing
step 4. $x_{ikt} > UM_{ik}$. If the condition is true $x_{ikt} \leftarrow UM_{ik}$, otherwise do nothing

Figure 21 – Pseudocode: repair function (condition 4)

4. Computational experiments

The computational experiments use the best four combinations of parameter values as discussed in section 3.4.4.. Since GAs are stochastic search heuristics different runs of the algorithm may lead to different "best" solutions, therefore each run of the proposed algorithm was repeated ten times. One run means running the algorithm for 52 weeks.

The problem instance used in these experiments was provided by a specific company. Decisions have to be made regarding the quantities to send from a single warehouse to 108 stores. There are eight products to be shipped on a weekly basis. Trucks transport the products from the warehouse to the stores. There are limits on transport quantities, store and warehouse capacities, that should not be exceeded. The decision to send the products is accompanied by handling, transportation and, eventually, holding costs. When the order to send a x quantities of products is decided, the warehouse has to organize the products (in pallets) and load the trucks – **warehouse handling costs**. The trucks then deliver the pallets to each respective store – **transportation costs**. The pallets are received by each store, verified, unboxed and properly stored – **store handling costs**. If the capacity of either the warehouse or any of the stores is exceeded, a cost per square meter (2 boxes) is incurred by the company – **holding costs**. The time horizon of the experiments is 52 weeks.

As we said before, we address the problem by solving each week in the isolation. However, since the decisions of one week impact the following week inventories, there is still some connection between the decisions. The quality of the solutions obtained, and in particular of the best one, will be compared with the current practice in the company. Comparisons will be made regarding weekly costs and accumulated costs along 52-weeks time horizon. Moreover, we will compare the costs disaggregated by source. The results used in these comparisons refer to the average values over the ten runs executed.

Figure 22 shows the weekly cumulative cost evolution along the 52-weeks. As it can be seen, the four combinations tested outperform current practice. This is particularly relevant towards the end of the time window. Our approach allows for an overall cost reduction of at least, on average, 6.60% (Combination 8), although with combination 4 this number goes up to 12.76%. In Figure 23 we can see that the current methods have

higher, weekly, costs up until week 23, it achieves better costs than Combination 10. In week 35 there is a big spike on the weekly cost (≈ 14,000%), which is the where, as seen in Figure 22, the costs of the current method really start to diverge from the all the combinations of the algorithm. However, the current methods start to converge in week 41, where, thereafter, the current methods start to obtain lower costs than the algorithm, as seen in Figure 23.

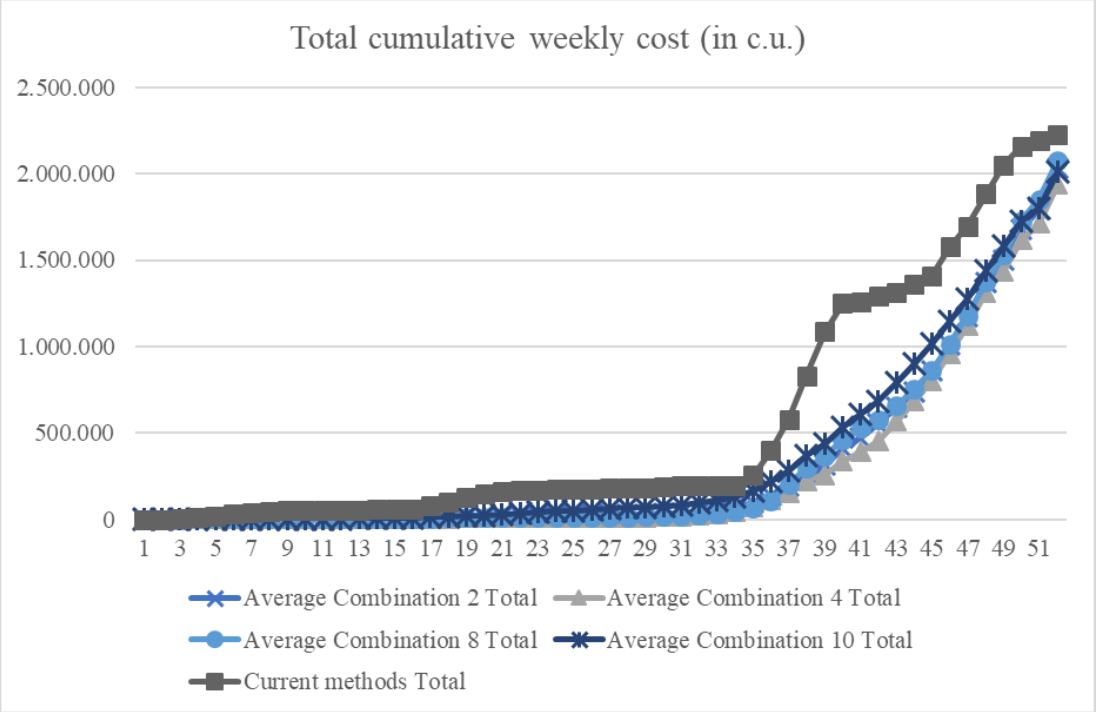


Figure 222 – Total cumulative weekly cost (in c.u.).

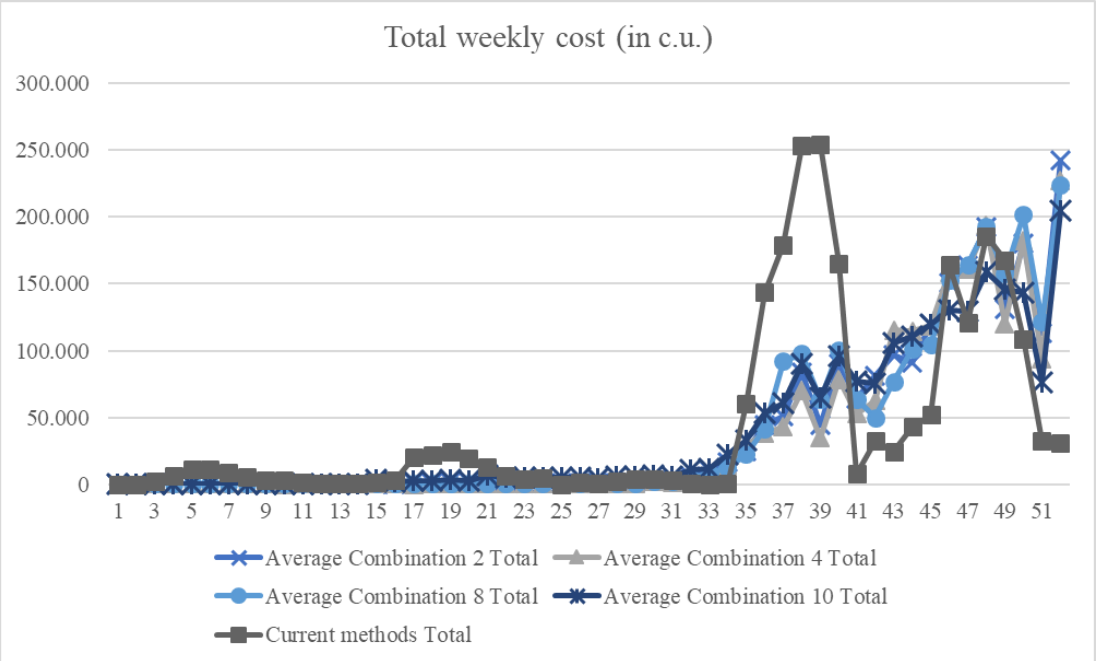


Figure 233 - Total weekly cost (in c.u.).

Full details of the costs for all 40 runs (ten runs of each of the four Combinations) runs can be found in Appendix 6 and in Appendix 7 the costs per Combination, per run. A summary of the results obtained with the five runs are shown in Table 6.

Combination no./Run no.	Total Cost (in c.u.)	Mean	Std. Dev.
Combination 2 Run 1	2,076,748	2,034,499	160,799
Combination 2 Run 2	1,911,035		
Combination 2 Run 3	2,057,506		
Combination 2 Run 4	2,119,819		
Combination 2 Run 5	1,810,830		
Combination 2 Run 6	2,165,034		
Combination 2 Run 7	1,879,739		
Combination 2 Run 8	**2,223,722**		
Combination 2 Run 9	1,849,593		
Combination 2 Run 10	**2,250,966**		
Combination 4 Run 1	1,727,164	1,938,824	301,809
Combination 4 Run 2	**2,509,217**		
Combination 4 Run 3	1,837,496		
Combination 4 Run 4	1,714,728		
Combination 4 Run 5	1,785,916		
Combination 4 Run 6	1,736,839		
Combination 4 Run 7	1,826,389		
Combination 4 Run 8	1,889,904		
Combination 4 Run 9	1,869,377		
Combination 4 Run 10	**2,491,209**		
Combination 8 Run 1	**2,227,859**	2,075,908	163,918
Combination 8 Run 2	2,075,910		
Combination 8 Run 3	2,203,699		
Combination 8 Run 4	1,801,760		
Combination 8 Run 5	1,983,953		
Combination 8 Run 6	2,198,894		
Combination 8 Run 7	2,034,424		
Combination 8 Run 8	**2,322,256**		
Combination 8 Run 9	1,877,394		
Combination 8 Run 10	2,032,927		
Combination 10 Run 1	2,208,788	2,007,966	266,491
Combination 10 Run 2	1,494,040		
Combination 10 Run 3	2,055,040		

Combination 10 Run 4	2,145,239		
Combination 10 Run 5	1,937,401		
Combination 10 Run 6	1,773,637		
Combination 10 Run 7	**2,275,324**		
Combination 10 Run 8	1,758,555		
Combination 10 Run 9	**2,345,559**		
Combination 10 Run 10	2,086,076		

Table 7 - Test results, means and standard deviations.

As it can be seen from Table 6, of the 40 runs executed only eight (in bold in the table) obtained a best solution having a total cost higher than that of the one used by the company.

Combination 4 yielded the best average result, but the largest variation. It finds some of the best results (second, third, and fourth best solutions amongst the 40 found); however, the best solution in its second run is the worst of them all, and the tenth is the second worst. Note that, this combination allows for an average improvement of 12.76% over current practices, ranging from an astonishing 22.84% to a loss of 12.90%. Therefore, this combination seems to be the less robust. Combination 2, on the other hand, seems to be the most robust one. Although on average it is only the third best out of four, it has the least variation. It doesn't always find a better solution than the current practices, however, it can improve the cost by 18.52%. The worst solution, produced by Combination 2, was 1.38% higher than the current methods (run number 10).

To analyze costs by sources average results (of the ten runs of each Combination) will be used. The operational costs (i.e., transportation and handling, both in the stores and warehouse) have a very small impact in the overall costs; they are always less than 1.29% in our results and about 0.95% in the current practice solution.

Figure 24 shows the storage costs incurred at the warehouse and at the stores. As it can be seen, for the current practice solution these costs are very similar; the ones incurred at the warehouse are slightly larger, accounting for about 52% of the total holding costs. Regarding the solution our method finds, the holding costs at the stores account for a huge part of the total holding costs, on average between 84% and 92%. This is easily explained given our approach. We have defined as a priority the non-violation of the warehouse capacity and whenever it would occur the repair mechanism of our algorithm "floods"

the stores with product. However, note that the violation of both the warehouse and store's capacity is inevitable since in the considered instance the inflow of stock (on order) is higher than the outflow (sales) – 9.3M units vs 7.5M units.

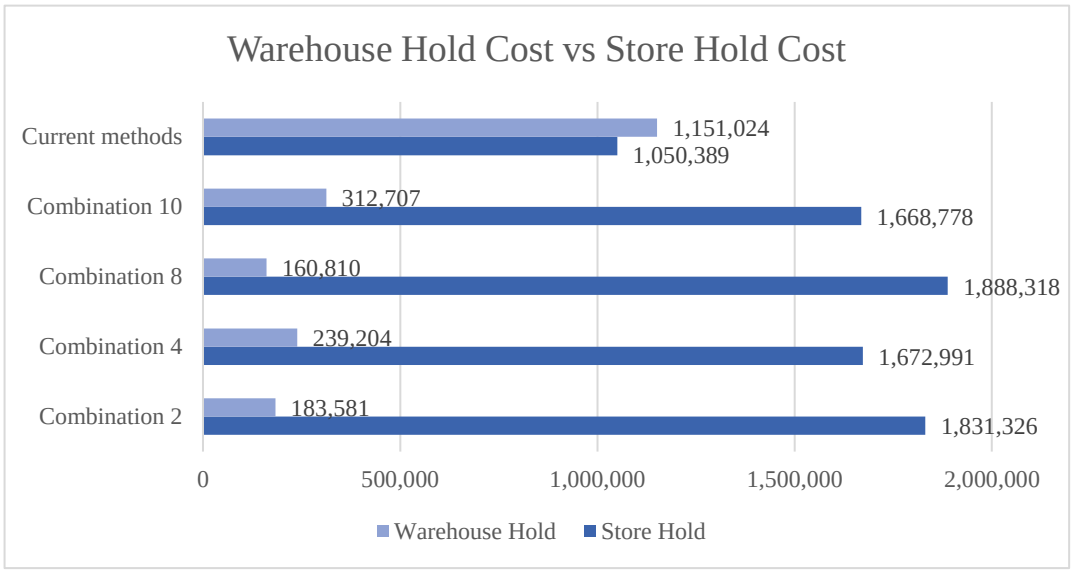


Figure 24 - Warehouse Hold Cost vs Store Hold Cost.

In Figure 25 we can see an example of the repair mechanism working, each time the warehouse stock exceeds the limit or is about to.

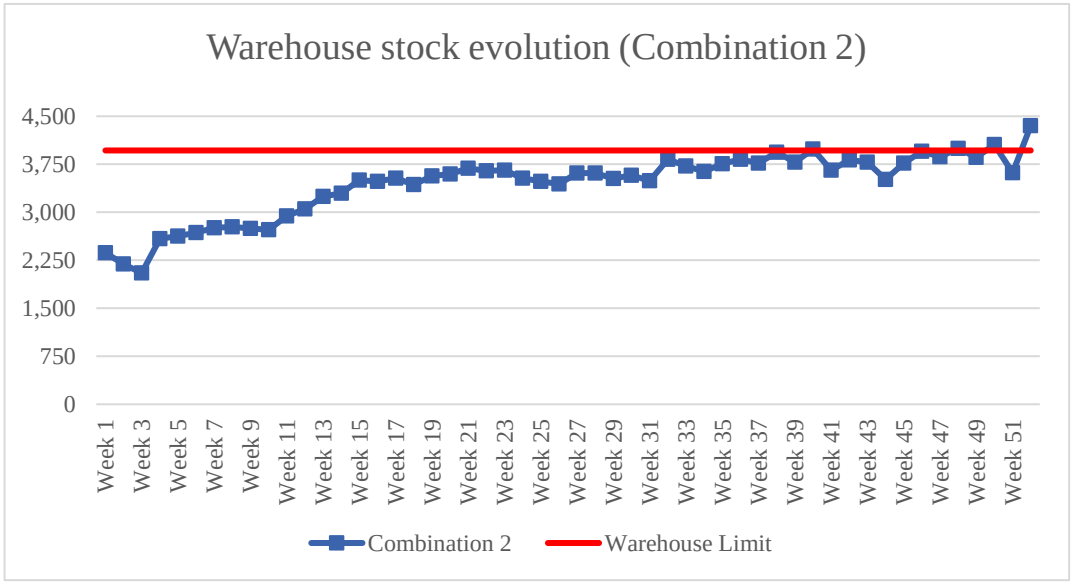


Figure 25 - Warehouse stock evolution example (Combination 2).

Finally, we will analyze runtimes. The algorithm was implemented in Python 3.6.0. The experiments were executed on a personal computer running Windows 10 64 bits on 2 2.2GHz Intel® Core™ i5-5200U CPUs with access to 6 Gb RAM.

With the current methods, a downstream manager spends 7.5 to 10 hours a week to analyze the quantities, of each product, to send to each store. Although, the number of products being analyzed by the manager is greater than the number that we used in our algorithm. A downstream manager looks at, at least, 30 products per season. It was not possible to access more data than what was used (i.e., 108 stores, 8 products, 52 weeks). Furthermore, the results can be used to compare with future work and to further convince the managers that the tool is a good option to what is used today. Table 7 gives us the runtime of each combination.

Combination.	Iterations per second	Generations	Time per iteration (in minutes[8])	Time to obtain best solution[9] (in hours[7])
2	2.85	200	1’10’’	1h00’
4	1.10	200	3’02’’	2h38’
8	1.31	200	2’33’’	2h12’
10	2.87	800	4’39’’	4h01’

Table 8 - Runtime per combination

To conclude the analysis of the results, we believe that Combination 2 is the most reliable. With the tests that were executed, Combination 2, despite not yielding the best results, has the lowest variance and runs faster than remaining combinations, providing a solution in one hour.

[8] (‘) represents minutes while (’’) represents seconds.
[9] Considering a time window of 52 weeks.

5. Conclusions

Genetic algorithms have been used with success in lot sizing problems (Kimms, 1999; Lee et al., 1997; Sikora, 1996; Xie & Dong, 2002). This dissertation addresses a specific problem within the lot sizing problems. Furthermore, it proposes a genetic algorithm to solve the problem as an alternative to the methods that are currently employed by a specific company.

We fully analyzed the problem and looked at all the variables that may influence the problem's outcome. We included all the problem variables that we deemed as important to the company and relevant for the business. It was a big challenge including variables that, normally, the managers do not use and which little to no data is available (e.g., holding and handling costs). Nevertheless, the assumptions that we considered were solid enough to overcome the issue.

Some modifications were made to the standard genetic algorithm procedure. Firstly, population is generated at random. However, the values of the chromosomes are limited by a lower and upper bound. We used the transportation quotas as limiting factors. By seeding this information we allow the individuals achieve good fitness scores in the early stages (Rajan & Shende, 1999; Casella & Potter, 2005)

The fitness function introduced is different than the standard. A score system was introduced. The individuals are given points not only for fulfillment of the objective function, but also for not violating constraints of the problem. Thus, admissibility of a solution can also be assessed throughout the execution of the algorithm. As far as we are aware of, this is proposed here for the first time and the computational experiments show it worked well in this problem. Nevertheless, a repair function is still used to rectify any deviations from the constraints of the problem. However, this is done only at the end of each full iteration, i.e., week, when the final solution is provided.

To select the parents to apply crossover we used an elitist approach. Chromosomes that have better scores are placed in a pool from which two are randomly pick for reproduction. Elitist is also to copy the best solutions to the next generation. Finally, mutation is introduced by resorting to mutants, which has never been done in the context

of lot sizing problems. This approach gives more relevance to mutation operator, since it is used in every generation, not at a low probability.

The algorithm is capable of answering to the problem at hands in a reasonable amount of time. The tests showed that the algorithm can produce good solutions in comparison to the methods currently used.

In the future, we aim at improving the algorithm's code, making it more Pythonic[10], and more efficient. Additionally, making the tool more user friendly and more functional, (e.g. reading information directly from an Excel file), and ultimately easier to adapt to new situations. Furthermore, we understand that the results need further testing. Considering that most of the time was dedicated to build (i.e., code) the algorithm, it left little time to test and further prove the consistency of the results. Finally, solution quality could benefit from an integrated approach considering all, or at least some of the periods simultaneously.

[10] "Exploiting the features of the Python language to produce code that is clear, concise and maintainable." from Stack Overflow (https://stackoverflow.com/questions/25011078/what-does-pythonic-mean)

6. Bibliography

Aggarwal, A., & Park, J. K. (1993). Improved algorithms for economic lot size problems. *Operations research, 41*(3), 549-571.

Altiparmak, F., Gen, M., Lin, L., & Paksoy, T. (2006). A genetic algorithm approach for multi-objective optimization of supply chain networks. *Computers & Industrial Engineering, 51*(1), 196-215. doi:10.1016/j.cie.2006.07.011

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA journal on computing, 6*(2), 154-160.

Beasley, D., Bull, D. R., & Martin, R. R. (1993). An overview of genetic algorithms: Part 1, fundamentals. *University computing, 15*(2), 56-69.

Ben-Daya, M., Darwish, M., & Ertogral, K. (2008). The joint economic lot sizing problem: Review and extensions. *European Journal of Operational Research, 185*(2), 726-742.

Bruce, M., Daly, L., & Towers, N. (2004). Lean or agile: A solution for supply chain management in the textiles and clothing industry? *International Journal of Operations & Production Management, 24*(2), 151-170. doi:10.1108/01443570410514867

Burke, E. K., Gustafson, S., & Kendall, G. (2004). Diversity in genetic programming: An analysis of measures and correlation with fitness. *IEEE Transactions on evolutionary computation, 8*(1), 47-62.

Casella, D. A., & Potter, W. D. (2005). *New Lower Bounds for the Snake-in-the-Box Problem: Using Evolutionary Techniques to Hunt for Snakes.* Paper presented at the FLAIRS Conference.

Deb, K., & Agrawal, S. (1998). *Understanding Interactions among Genetic Algorithm Parameters.* Paper presented at the FOGA.

Decreto-Lei n.º 86-B/2016, de 29 de dezembro, www.dre.pt, Diário da República, 1.ª série — N.º 249 Stat. (2016).

Dellaert, N., Jeunet, J., & Jonard, N. (2000). A genetic algorithm to solve the general multi-level lot-sizing problem with time-varying costs. *International Journal of Production Economics, 68*(3), 241-257.

Diaz-Gomez, P. A., & Hougen, D. F. (2007). *Initial Population for Genetic Algorithms: A Metric Approach.* Paper presented at the GEM.

Du Merle, O., Goffin, J.-L., Trouiller, C., & Vial, J.-P. (2000). A lagrangian relaxation of the capacitated multi-item lot sizing problem solved with an interior point cutting plane algorithm *Approximation and Complexity in Numerical Optimization* (pp. 380-405): Springer.

Eiben, Á. E., Hinterding, R., & Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. *IEEE Transactions on evolutionary computation, 3*(2), 124-141.

Elmaghraby, S. E. (1978). The economic lot scheduling problem (ELSP): review and extensions. *Management science, 24*(6), 587-598.

Federgruen, A., & Tzur, M. (1991). A simple forward algorithm to solve general dynamic lot sizing models with n periods in 0 (n log n) or 0 (n) time. *Management science, 37*(8), 909-925.

Gaafar, L. (2006). Applying genetic algorithms to dynamic lot sizing with batch ordering. *Computers & Industrial Engineering, 51*(3), 433-444. doi:10.1016/j.cie.2006.08.006

Gen, M., Altiparmak, F., & Lin, L. (2006). A genetic algorithm for two-stage transportation problem using priority-based encoding. *OR Spectrum, 28*(3), 337-354.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*: Addison-Wesley Longman Publishing Co., Inc.

Goldberg, D. E. (2006). *Genetic algorithms*: Pearson Education India.

Grefenstette, J. J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on systems, man, and cybernetics, 16*(1), 122-128.

Grefenstette, J. J., & Baker, J. E. (1989). *How genetic algorithms work: A critical look at implicit parallelism.* Paper presented at the Proceedings of the third international conference on Genetic algorithms.

Hanssmann, F. (1962). *Operations Research in Production and Inventory Control*: Wiley.

Haupt, R. L. (1995). An introduction to genetic algorithms for electromagnetics. *IEEE Antennas and Propagation Magazine, 37*(2), 7-15.

Haupt, R. L., & Haupt, S. E. (2004). *Practical genetic algorithms*: John Wiley & Sons.

Holland, J. H. (1975). Adaptation in natural and artificial systems. An introductory analysis with application to biology, control, and artificial intelligence. *Ann Arbor, MI: University of Michigan Press*.

Hsu, W.-L. (1983). On the general feasibility test of scheduling lot sizes for several products on one machine. *Management science, 29*(1), 93-105.

Jans, R., & Degraeve, Z. (2004). Meta-heuristics for dynamic lot sizing: a review and comparison of solution approaches.

Jong, K. A. D. (1975). *An analysis of the behavior of a class of genetic adaptive systems.* University of Michigan.

Kaminsky, P., & Simchi-Levi, D. (2003). Production and Distribution Lot Sizing in a Two Stage Supply Chain. *IIE Transactions, 35*(11), 1065-1075. doi:10.1080/07408170304401

Karimi, B., Ghomi, S. F., & Wilson, J. (2003). The capacitated lot sizing problem: a review of models and algorithms. *Omega, 31*(5), 365-378.

Khouja, M., Michalewicz, Z., & Wilmot, M. (1998). The use of genetic algorithms to solve the economic lot size scheduling problem. *European Journal of Operational Research, 110*(3), 509-524.

Kimms, A. (1999). A genetic algorithm for multi-level, multi-machine lot sizing and scheduling. *Computers & Operations Research, 26*(8), 829-848.

Lee, I., Sikora, R., & Shaw, M. J. (1997). A genetic algorithm-based approach to flexible flow-line scheduling with variable lot sizes. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 27*(1), 36-54.

Leung, Y., Gao, Y., & Xu, Z.-B. (1997). Degree of population diversity-a perspective on premature convergence in genetic algorithms and its markov chain analysis. *IEEE Transactions on Neural Networks, 8*(5), 1165-1176.

Lobo, F. G., & Lima, C. F. (2005). *A review of adaptive population sizing schemes in genetic algorithms.* Paper presented at the Proceedings of the 7th annual workshop on Genetic and evolutionary computation.

Melanie, M. (1999). An introduction to genetic algorithms. *Cambridge, Massachusetts London, England, Fifth printing, 3*, 62-75.

Merelo, J. J., & Prieto, A. (1996). *GAGS, a flexible object-oriented library for evolutionary computation.* Paper presented at the Proc. of the First International

Workshop on Machine learning, Forecasting and Optimization (MALFO'96), ISBN.

Portaria n.º 156/2014. (156/2014). (2014).

Rajan, D. S., & Shende, A. M. (1999). Maximal and reversible snakes in hypercubes.

Rogers, J. (1958). A computational approach to the economic lot scheduling problem. *Management science, 4*(3), 264-291.

Rudolph, G. (2001). Evolutionary search under partially ordered fitness sets. *HT014601767*.

Sarker, R., & Newton, C. (2002). A genetic algorithm for solving economic lot size scheduling problem. *Computers & Industrial Engineering, 42*(2), 189-198.

Sikora, R. (1996). A genetic algorithm for integrating lot-sizing and sequencing in scheduling a capacitated flow line. *Computers & Industrial Engineering, 30*(4), 969-981.

Silver, E. A., & Meal, H. C. (1973). A heuristic for selecting lot size quantities for the case of a deterministic time-varying demand rate and discrete opportunities for replenishment. *Production and inventory management, 14*(2), 64-74.

Toso, R. F., & Resende, M. G. (2015). A C++ application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software, 30*(1), 81-93.

Wagner, H. M., & Whitin, T. M. (1958). Dynamic version of the economic lot size model. *Management science, 5*(1), 89-96.

Wee, H.-M., & Shum, Y.-S. (1999). Model development for deteriorating inventory in material requirement planning systems. *Computers & Industrial Engineering, 36*(1), 219-225.

Xie, J., & Dong, J. (2002). Heuristic genetic algorithms for general capacitated lot-sizing problems. *Computers & Mathematics with applications, 44*(1-2), 263-276.

Zangwill, W. I. (1969). A backlogging model and a multi-echelon model of a dynamic economic lot size production system—a network approach. *Management science, 15*(9), 506-527.

Zitzler, E., Deb, K., & Thiele, L. (2000). Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation, 8*(2), 173-195.

7. Appendix

7.1. Appendix 1

Cost per zone:

Zone	Cost per m2
z1	679,35
z2	602,92
z3	557,91

Zone municipality allocations:

Municipality	Zone
Regiões Autónomas	z1
Almada	z1
Amadora	z1
Barreiro	z1
Cascais	z1
Gondomar	z1
Loures	z1
Maia	z1
Matosinhos	z1
Moita	z1
Montijo	z1
Odivelas	z1
Oeiras	z1
Póvoa do Varzim	z1
Seixal	z1
Sintra	z1
Valongo	z1
Vila do Conde	z1
Vila Franca de Xira	z1
Vila Nova de Gaia	z1
Abrantes	z2
Albufeira	z2
Alenquer	z2
Caldas da Rainha	z2
Chaves	z2

Municipality	Zone
Covilhã	z2
Elvas	z2
Entroncamento	z2
Espinho	z2
Estremoz	z2
Figueira da Foz	z2
Guimarães	z2
Ílhavo	z2
Lagos	z2
Loulé	z2
Olhão	z2
Palmela	z2
Peniche	z2
Peso da Régua	z2
Portimão	z2
Santiago do Cacém	z2
São João da Madeira	z2
Sesimbra	z2
Silves	z2
Sines	z2
Tomar	z2
Torres Novas	z2
Torres Vedras	z2
Vila Real de Santo António	z2
Vizela	z2

7.2. Appendix 2

Transportation: products lower bound limit:

Store	Woman Apparel	Woman Underwear	Woman Footwear	Woman Accessories	Man Apparel	Man Underwear	Man Footwear	Man Accessories
1	160	7	2	3	219	76	2	4
2	111	4	3	4	31	27	1	2
3	15	2	1	2	12	9	1	1
4	123	7	1	6	163	16	1	3
5	33	42	1	8	68	28	1	2
6	17	5	1	2	52	12	1	1
7	53	48	3	5	21	3	1	7
8	7	15	1	2	2	7	1	1
9	29	2	1	2	33	1	1	1
10	42	5	1	3	111	3	1	1
11	47	24	2	11	218	6	2	5
12	71	27	2	3	88	24	1	2
13	22	11	1	1	19	11	1	2
14	19	7	1	5	67	21	1	2
15	146	8	1	5	136	4	2	2
16	14	13	2	3	4	3	1	1
17	8	20	1	4	175	3	1	3
18	122	11	4	4	201	62	1	3
19	47	4	2	1	2	15	1	4
20	221	8	3	11	295	6	1	9
21	224	3	3	3	154	26	1	3
22	86	1	1	3	14	3	1	1
23	132	12	1	12	5	3	1	6
24	102	11	3	1	59	35	1	3
25	14	15	2	7	78	6	2	1
26	118	7	3	14	74	3	1	5
27	188	3	1	3	10	16	1	1
28	85	47	1	1	59	35	1	3
29	119	25	1	2	84	19	1	2
30	8	1	2	6	1	3	1	1
31	49	8	1	3	45	9	1	2
32	71	35	1	7	62	30	1	3
33	12	11	2	4	7	2	1	2

34	1	3	1	1	10	3	1	1
35	41	5	1	8	127	5	1	5
36	136	5	1	2	116	3	1	2
37	140	4	2	4	91	3	2	5
38	223	12	3	2	122	33	1	3
39	55	9	1	5	24	16	1	1
40	83	12	1	3	67	32	1	1
41	1	8	1	4	3	2	1	1
42	175	11	1	2	176	3	1	3
43	201	7	2	15	92	53	1	5
44	49	1	1	2	95	3	1	2
45	28	4	1	1	45	37	1	1
46	123	5	3	4	51	37	2	3
47	247	2	1	4	215	3	2	3
48	40	5	1	2	60	2	1	1
49	20	4	1	2	43	11	1	1
50	65	21	3	4	145	11	1	1
51	1	11	1	3	73	6	1	2
52	5	2	1	1	1	2	1	1
53	30	2	1	4	14	6	1	1
54	118	17	1	14	109	39	1	3
55	36	10	1	10	158	37	1	2
56	50	5	2	1	49	5	1	3
57	87	4	1	4	72	6	1	1
58	94	3	1	1	7	3	1	1
59	16	7	2	11	39	43	1	3
60	62	10	1	2	134	25	1	2
61	21	7	1	4	24	5	1	1
62	2	12	1	3	44	3	1	3
63	78	17	1	4	11	2	1	1
64	177	2	1	2	119	20	1	2
65	196	6	2	3	30	3	1	3
66	27	12	1	1	91	3	1	1
67	10	2	1	1	1	3	1	1
68	21	8	2	4	22	22	1	1
69	78	1	1	3	47	3	1	1
70	38	4	1	2	66	3	1	1
71	1	7	2	2	123	4	1	3
72	91	43	2	8	87	35	1	2

73	51	13	1	1	14	10	1	1
74	28	9	1	8	18	1	1	5
75	46	9	3	2	114	3	1	1
76	2	6	1	4	2	2	1	1
77	39	16	3	8	105	31	2	2
78	59	12	1	1	54	7	1	2
79	59	11	2	2	90	4	1	1
80	158	9	2	8	11	21	1	2
81	73	10	1	7	118	20	1	3
82	11	52	1	3	56	25	1	4
83	20	4	1	3	59	5	1	2
84	124	81	1	11	126	26	1	2
85	3	1	1	1	7	1	1	1
86	10	3	1	1	48	2	1	1
87	41	18	1	2	72	1	1	1
88	4	13	1	7	29	3	1	2
89	13	8	1	8	59	3	1	6
90	72	6	1	2	23	1	1	1
91	1	1	2	2	1	1	1	1
92	22	9	1	1	18	2	1	2
93	127	46	1	4	256	3	1	1
94	41	7	2	13	92	52	1	1
95	1	1	1	1	7	1	1	1
96	285	2	3	3	17	12	4	11
97	57	9	4	4	80	16	1	3
98	14	6	1	3	1	21	1	1
99	63	11	2	2	37	3	1	2
100	33	5	3	3	41	23	1	6
101	1	3	6	1	66	3	1	2
102	98	9	1	6	51	9	1	1
103	1	10	3	2	3	3	2	2
104	179	5	5	2	54	12	1	1
105	10	3	1	3	1	3	1	1
106	197	5	1	3	115	29	1	3
107	20	3	1	4	7	2	1	2
108	58	81	1	7	144	56	2	3

Transportation: products upper bound limit:

Store	Woman Apparel	Woman Underwear	Woman Footwear	Woman Accessories	Man Apparel	Man Underwear	Man Footwear	Man Accessories
1	3415	984	308	804	1395	638	202	164
2	1172	675	267	581	886	460	229	100
3	838	600	294	420	664	470	200	79
4	1213	543	269	458	966	325	190	105
5	951	540	297	561	705	418	226	111
6	592	426	244	361	541	326	132	99
7	1889	798	281	1183	1206	457	381	154
8	1138	921	159	729	981	552	407	127
9	1023	646	244	550	565	257	150	90
10	2145	820	258	473	1136	510	212	102
11	2237	1166	279	830	1589	636	192	113
12	1095	633	200	714	641	362	217	96
13	1006	881	273	361	559	461	150	123
14	914	581	296	591	586	400	230	90
15	1152	616	259	541	880	344	231	92
16	909	1011	290	465	861	744	197	153
17	1726	801	261	639	1482	510	191	122
18	2264	895	303	843	1467	591	190	143
19	1287	647	303	433	866	406	187	127
20	2639	1379	280	724	2114	810	198	160
21	2091	662	266	700	1282	471	200	184
22	975	651	297	374	691	415	198	73
23	2203	855	251	786	1344	537	193	136
24	1185	730	257	654	1237	469	190	146
25	2456	840	254	628	1255	408	192	110
26	2257	826	264	628	1737	629	200	146
27	2335	660	281	831	1422	329	153	156
28	1164	624	249	538	815	553	151	95
29	1146	471	263	580	929	288	229	72
30	1920	923	297	648	1469	382	257	173
31	890	419	243	574	1010	348	122	97
32	1348	838	253	468	932	412	200	83
33	893	569	308	662	586	477	153	107
34	1613	2017	205	664	920	647	207	231
35	1239	712	256	479	1119	582	229	183

36	1387	770	269	662	1133	447	227	180
37	2819	1104	259	546	1281	748	235	148
38	2262	766	256	712	1353	361	192	110
39	1359	888	252	567	819	276	204	115
40	1494	670	260	385	1091	590	223	117
41	872	714	271	418	996	314	225	98
42	1791	651	282	884	1024	333	191	159
43	2360	942	259	637	964	439	188	172
44	1035	608	302	462	831	396	229	76
45	903	592	284	469	695	500	192	85
46	1418	646	250	527	851	369	196	88
47	2233	562	272	665	1512	436	197	116
48	886	687	284	348	661	351	206	90
49	975	521	299	446	659	345	229	116
50	1177	1010	228	533	877	500	227	86
51	1066	874	202	522	910	583	225	109
52	3296	1017	281	414	1872	582	339	201
53	957	655	302	469	700	365	233	78
54	2163	784	261	711	1156	514	229	120
55	1242	598	194	771	1245	339	213	91
56	1188	1121	314	514	1212	513	196	151
57	947	632	301	586	649	395	229	81
58	988	842	307	459	733	420	204	114
59	1558	780	265	662	1030	447	185	181
60	1177	777	258	713	943	506	224	78
61	912	799	284	413	521	501	155	99
62	1307	881	253	597	670	483	227	103
63	1432	730	261	505	937	344	242	121
64	1317	701	259	705	717	439	191	110
65	2362	735	294	636	1514	507	189	184
66	1389	674	254	660	1032	632	224	134
67	1062	937	258	577	1348	718	204	117
68	957	593	296	500	622	518	158	100
69	977	841	312	561	656	413	204	95
70	990	606	289	453	655	384	226	85
71	1128	446	244	377	806	311	219	109
72	2150	723	265	659	1278	333	230	90
73	1318	584	254	570	663	203	216	104
74	1425	1037	271	630	1113	497	205	157

75	1211	1091	202	322	1336	941	238	122
76	1487	1011	326	576	1991	351	291	153
77	1611	802	266	634	1147	372	196	108
78	928	826	297	393	819	630	224	105
79	1226	655	253	502	794	488	152	85
80	2109	854	246	735	1427	533	228	119
81	1377	643	244	689	966	531	212	98
82	1267	681	264	733	835	374	193	83
83	1125	754	276	579	853	549	208	99
84	1544	811	249	725	836	414	228	104
85	4236	938	335	993	1132	763	345	140
86	1163	630	197	410	571	202	183	86
87	1001	505	196	772	824	327	242	125
88	969	584	259	473	547	304	150	74
89	1300	841	242	743	1059	637	231	133
90	1168	410	173	533	716	304	233	80
91	1871	510	318	396	2190	714	431	306
92	1134	495	153	455	477	223	206	61
93	2473	1259	245	691	2540	1025	190	116
94	2621	912	266	475	1114	550	182	151
95	2274	880	301	494	1715	379	572	396
96	2052	1275	341	769	2024	616	464	176
97	995	578	227	512	752	445	219	89
98	812	605	158	856	1533	220	191	109
99	1365	667	180	789	821	470	240	123
100	2746	982	801	740	2166	971	544	267
101	1516	435	137	309	807	405	237	69
102	889	345	197	444	796	235	191	98
103	1363	746	242	439	785	473	245	86
104	2161	812	231	681	1383	981	248	159
105	1931	793	336	341	2017	820	167	209
106	1251	586	252	634	912	346	170	236
107	798	567	132	290	1019	327	162	98
108	1975	1206	90	415	1825	755	145	87

7.3. Appendix 3

The 27 possible parameter combinations:

Test no.	Generations	Population	Crossover	Elite	Mutants
1	200	30	0.75	0.15	0.10
2	200	30	0.79	0.20	0.01
3	200	30	0.80	0.15	0.05
4	200	40	0.75	0.15	0.10
5	200	40	0.79	0.20	0.01
6	200	40	0.80	0.15	0.05
7	200	60	0.75	0.15	0.10
8	200	60	0.79	0.20	0.01
9	200	60	0.80	0.15	0.05
10	800	30	0.75	0.15	0.10
11	800	30	0.79	0.20	0.01
12	800	30	0.80	0.15	0.05
13	800	40	0.75	0.15	0.10
14	800	40	0.79	0.20	0.01
15	800	40	0.80	0.15	0.05
16	800	60	0.75	0.15	0.10
17	800	60	0.79	0.20	0.01
18	800	60	0.80	0.15	0.05
19	2,000	30	0.75	0.15	0.10
20	2,000	30	0.79	0.20	0.01
21	2,000	30	0.80	0.15	0.05
22	2,000	40	0.75	0.15	0.10
23	2,000	40	0.79	0.20	0.01
24	2,000	40	0.80	0.15	0.05
25	2,000	60	0.75	0.15	0.10
26	2,000	60	0.79	0.20	0.01
27	2,000	60	0.80	0.15	0.05

7.4. Appendix 4

Test results per week, per test: tests 1-9:

Test no.	1	2	3	4	5	6	7	8	9
Week 1	469	370	413	409	361	422	390	384	351
Week 2	460	377	395	423	362	406	400	349	343
Week 3	468	394	450	458	431	467	418	452	517
Week 4	485	378	492	450	418	462	423	430	474
Week 5	564	444	434	450	423	426	437	411	403
Week 6	570	434	471	473	441	431	539	418	450
Week 7	508	451	433	457	433	436	513	444	409
Week 8	446	427	497	302	379	535	541	474	532
Week 9	345	585	567	621	582	597	590	628	571
Week 10	519	367	396	324	316	374	337	448	381
Week 11	337	554	557	458	471	547	647	541	535
Week 12	501	423	457	490	438	418	462	434	482
Week 13	527	565	535	525	431	545	532	491	549
Week 14	438	363	386	453	379	420	473	400	347
Week 15	403	527	473	495	491	373	340	491	513
Week 16	275	434	399	392	437	398	426	417	393
Week 17	455	381	368	400	2.451	385	391	392	391
Week 18	348	421	396	380	1.860	410	398	403	379
Week 19	551	493	468	479	1.975	447	482	476	442
Week 20	489	471	467	451	1.926	460	447	453	455
Week 21	467	466	412	434	1.985	469	466	474	437
Week 22	500	496	508	452	1.976	471	458	452	461
Week 23	490	452	484	503	1.980	526	502	452	514
Week 24	560	536	500	485	1.980	477	533	529	490
Week 25	531	521	519	477	2.034	529	520	496	537

Test results per week, per test: tests 10-18:

Test no.	10	11	12	13	14	15	16	17	18
Week 1	362	324	335	326	328	326	315	328	325
Week 2	347	323	326	336	338	333	326	326	325
Week 3	448	349	380	384	320	341	424	349	347
Week 4	426	321	394	395	340	422	462	330	376
Week 5	453	377	364	370	374	362	358	356	344

Week 6	428	364	378	416	393	376	366	389	342
Week 7	442	394	361	371	410	378	365	379	348
Week 8	341	363	312	303	409	372	279	368	354
Week 9	325	548	406	335	412	430	365	325	401
Week 10	400	391	430	328	443	376	303	406	361
Week 11	428	513	463	403	346	383	406	406	457
Week 12	464	393	404	456	414	385	434	408	365
Week 13	405	539	349	371	1,668	363	2,767	1,308	613
Week 14	419	351	358	374	1,248	359	2,746	1,139	2,842
Week 15	503	1,504	467	6,259	4,754	3,870	4,974	6,253	8,167
Week 16	477	1,201	425	493	4,033	1,314	14,251	5,226	6,890
Week 17	436	2,548	537	6,737	6,825	1,295	14,440	10,699	8,930
Week 18	482	1,991	1,261	2,672	6,020	1,366	17,546	9,452	7,493
Week 19	476	4,550	1,281	2,725	8,767	4,251	20,538	14,295	9,611
Week 20	515	3,465	1,322	2,735	8,168	2,726	22,001	13,006	25,556
Week 21	491	7,487	2,468	17,280	10,291	22,225	24,679	15,554	22,826
Week 22	469	6,355	2,205	4,765	8,901	3,863	21,808	14,356	18,057
Week 23	516	5,986	2,314	4,777	8,380	3,896	24,931	13,753	14,869
Week 24	508	5,563	2,294	4,764	7,796	3,928	22,160	13,172	13,432
Week 25	554	8,708	2,309	4,848	7,461	3,971	25,305	12,739	18,509

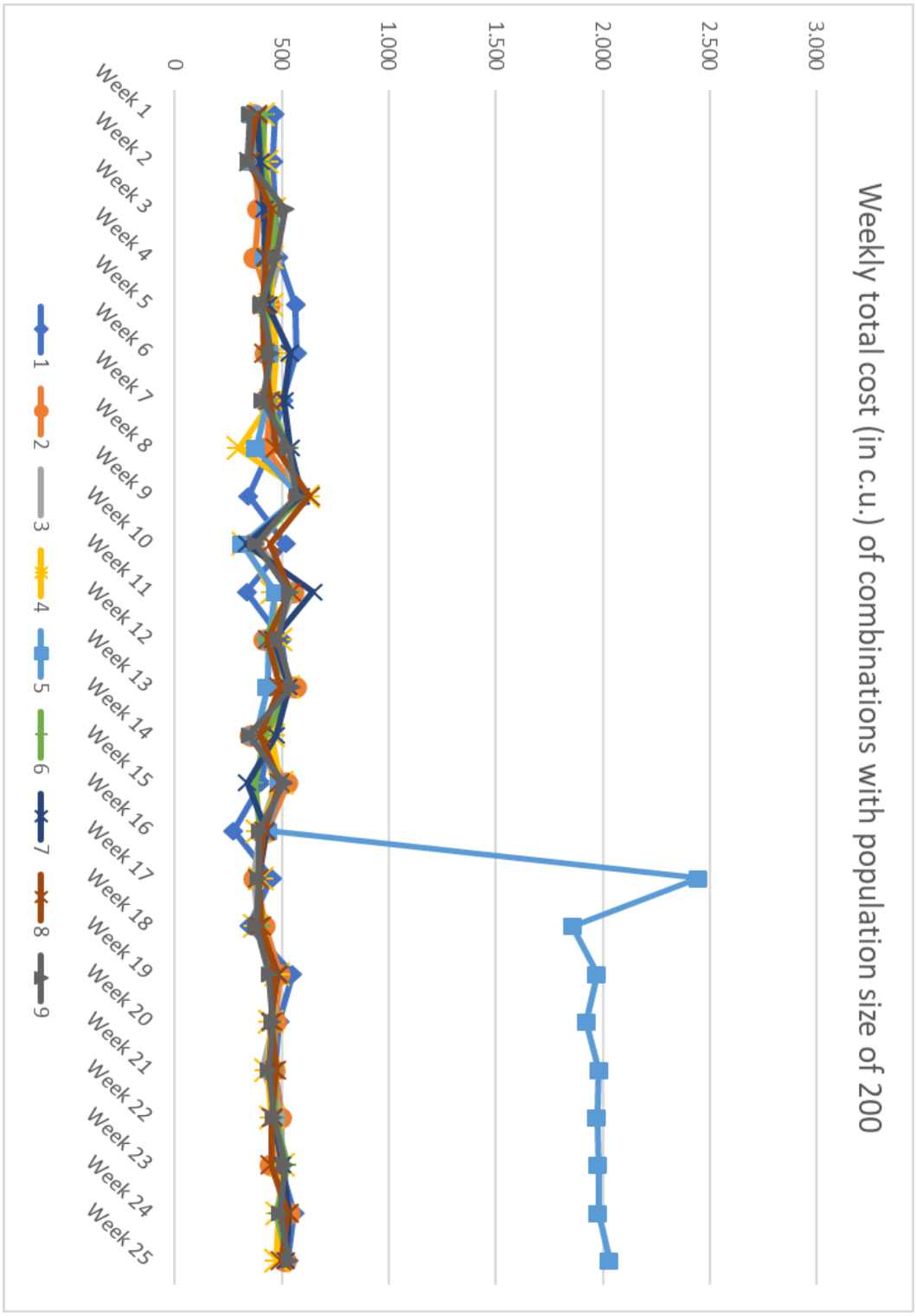
Test results per week, per test: tests 19-27:

Test no.	19	20	21	22	23	24	25	26	27
Week 1	321	314	310	315	323	313	316	333	314
Week 2	316	318	315	313	325	319	323	322	312
Week 3	326	319	327	317	321	320	320	319	320
Week 4	468	314	434	325	317	316	355	323	335
Week 5	339	348	343	336	351	338	334	339	343
Week 6	350	351	351	344	334	342	337	344	336
Week 7	393	353	358	339	350	341	358	354	346
Week 8	268	319	349	318	356	312	367	354	328
Week 9	573	364	484	452	371	393	561	529	366
Week 10	302	340	350	442	415	375	327	361	331
Week 11	417	325	407	384	340	409	554	367	402
Week 12	426	384	376	407	348	360	386	362	366
Week 13	1,733	2,408	2,368	1,243	1,304	2,248	539	1,439	1,869
Week 14	1,683	5,658	1,971	3,827	4,181	6,396	329	1,205	5,472
Week 15	3,718	13,253	4,883	6,159	8,286	10,566	2,271	3,097	13,222

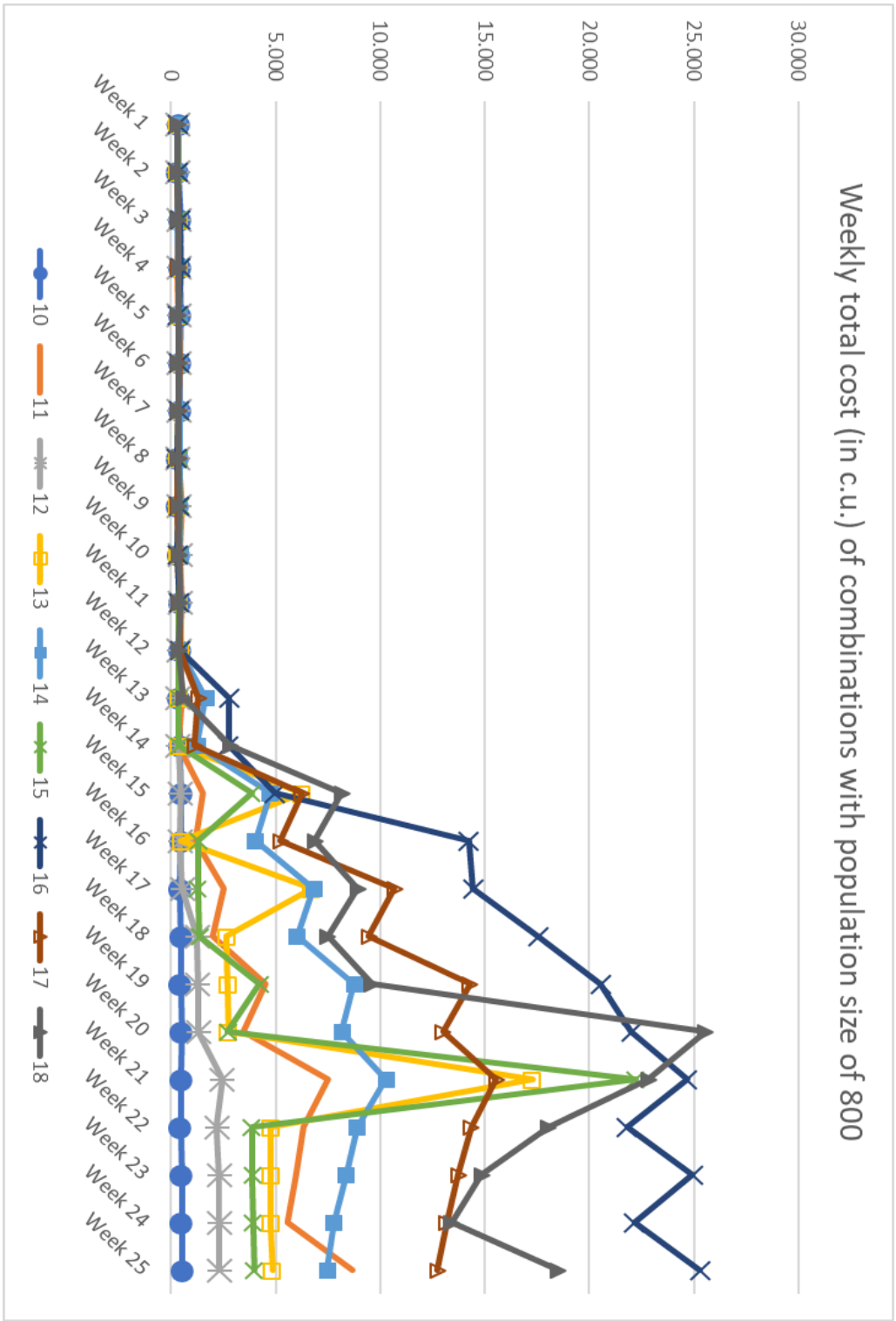
Week 16	4,844	10,232	4,394	18,676	7,041	9,900	1,947	2,606	11,713
Week 17	6,036	15,477	6,735	19,790	9,154	11,785	3,180	4,177	12,627
Week 18	7,356	13,247	5,688	21,394	7,190	11,026	3,109	8,508	11,813
Week 19	8,511	16,709	8,153	-	9,191	-	3,532	10,151	-
Week 20	9,390	14,943	11,814	-	8,133	-	5,498	8,485	-
Week 21	27,679	16,611	12,414	-	10,291	-	5,828	10,825	-
Week 22	13,131	15,220	10,416	-	-	-	6,280	-	-
Week 23	13,461	14,081	13,689	-	-	-	7,597	-	-
Week 24	15,911	13,501	11,516	-	-	-	8,818	-	-
Week 25	18,264	14,565	13,365	-	-	-	9,689	-	-

7.5. Appendix 5

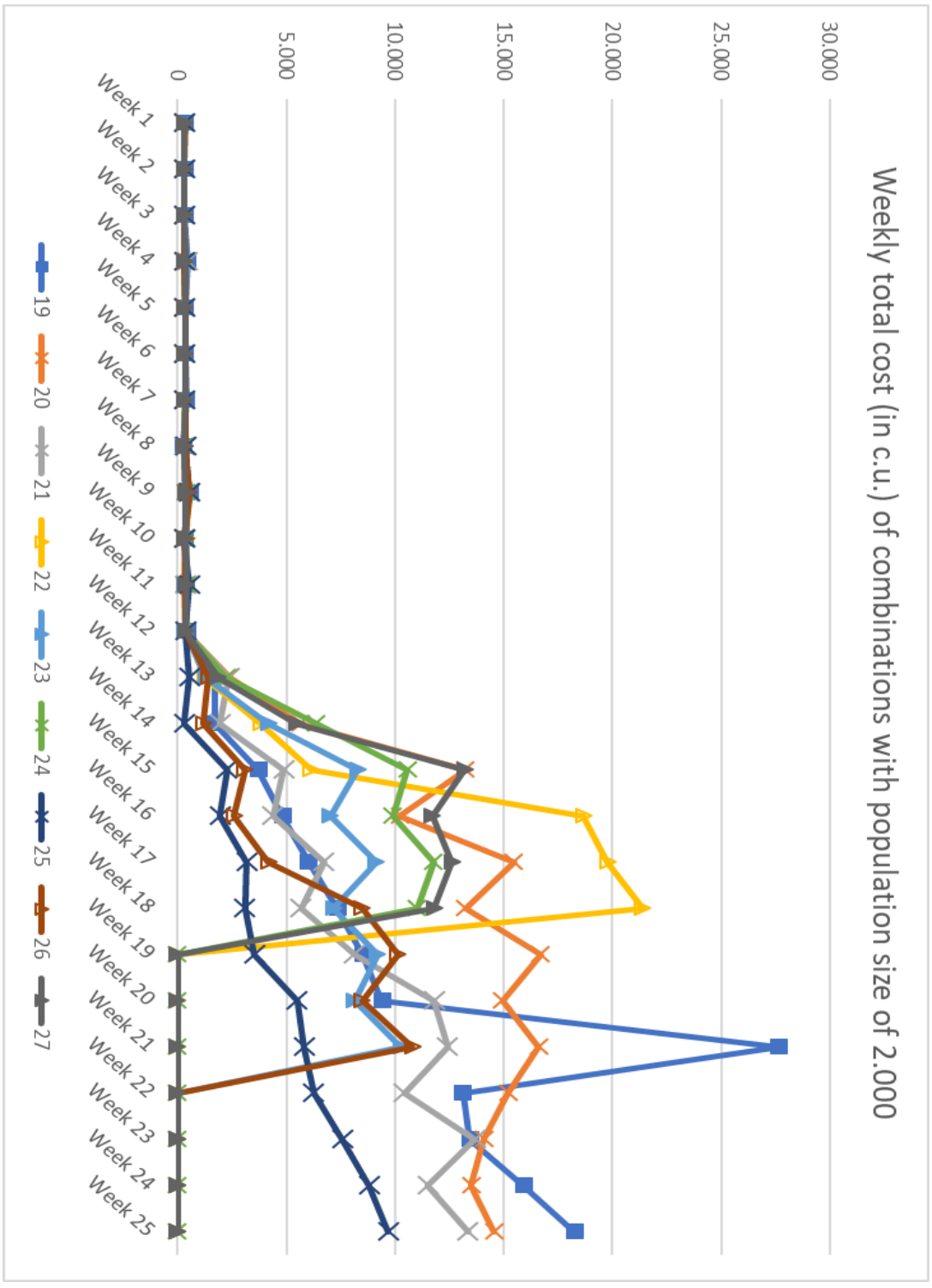
Weekly total cost (in c.u.) of combinations with population size 200:



Weekly total cost (in c.u.) of combinations with population size 800:



Weekly total cost (in c.u.) of combinations with population size 2,000:



7.6. Appendix 6

Average weekly cost of each combination and current methods:

Week	Combination 2	Combination 4	Combination 8	Combination 10	Current methods
Week 1	388	424	358	368	18
Week 2	393	425	367	364	23
Week 3	401	422	431	449	2,630
Week 4	404	450	435	449	6,726
Week 5	441	468	411	394	11,342
Week 6	434	482	429	403	11,716
Week 7	441	497	439	411	9,196
Week 8	420	499	440	330	5,483
Week 9	548	612	556	415	3,389
Week 10	411	342	436	382	2,999
Week 11	517	544	546	450	1,538
Week 12	433	491	447	448	455
Week 13	518	510	520	437	414
Week 14	389	412	401	391	411
Week 15	766	408	465	3,723	1,336
Week 16	672	428	413	2,344	3,111
Week 17	2,119	389	684	2,938	20,319
Week 18	1,315	403	606	2,530	21,865
Week 19	2,418	488	603	3,254	24,586
Week 20	1,791	475	581	3,020	19,714
Week 21	2,484	457	658	7,099	12,955
Week 22	1,847	486	640	5,069	6,475
Week 23	1,615	491	656	5,607	4,154
Week 24	1,516	527	682	4,927	5,049
Week 25	1,513	537	697	5,310	368
Week 26	1,495	494	674	4,929	1,798
Week 27	1,516	522	687	4,712	641
Week 28	1,518	536	700	6,129	2,299
Week 29	1,530	537	700	5,741	4,439
Week 30	3,218	2,404	2,603	6,781	4,126
Week 31	2,618	1,754	2,205	6,509	3,468
Week 32	5,804	4,639	4,089	10,984	1,094
Week 33	10,756	6,367	8,229	11,568	323
Week 34	16,330	14,320	12,893	22,744	418

Week 35	24,714	28,832	22,755	33,321	60,155
Week 36	44,605	38,787	41,708	53,443	143,447
Week 37	51,123	43,180	92,090	61,249	179,387
Week 38	83,740	70,261	97,932	89,948	253,278
Week 39	44,269	35,101	63,394	65,296	254,091
Week 40	90,166	78,812	100,835	95,648	165,427
Week 41	63,778	52,756	64,082	76,899	7,957
Week 42	81,092	62,713	49,854	75,483	32,454
Week 43	96,959	115,259	77,212	105,628	24,598
Week 44	90,781	114,529	101,272	110,494	43,256
Week 45	119,989	118,633	104,428	119,649	52,095
Week 46	150,784	158,657	152,574	129,993	164,162
Week 47	163,950	161,038	164,646	129,713	121,353
Week 48	192,373	193,647	192,634	158,731	185,734
Week 49	131,299	120,219	157,642	146,124	167,772
Week 50	180,007	182,086	201,667	143,332	108,987
Week 51	113,313	94,044	121,721	76,619	32,534
Week 52	242,578	227,032	223,782	204,785	30,952
Total	2,034,499	1,938,824	2,075,908	2,007,966	2,222,516

7.7. Appendix 7

Weekly cost of each run with combination 2 (run 1-5):

Week	Combination 2 Run 1	Combination 2 Run 2	Combination 2 Run 3	Combination 2 Run 4	Combination 2 Run 5
Week 1	372	380	375	410	382
Week 2	394	362	413	395	395
Week 3	406	420	414	376	396
Week 4	400	412	405	402	410
Week 5	431	435	472	446	428
Week 6	416	427	453	445	431
Week 7	415	445	434	442	447
Week 8	368	468	432	453	467
Week 9	579	544	567	578	559
Week 10	415	399	453	434	427
Week 11	539	547	528	546	561
Week 12	422	444	444	431	415
Week 13	547	513	539	505	592
Week 14	394	389	370	379	378
Week 15	452	391	485	494	495
Week 16	454	449	411	429	414
Week 17	403	407	378	416	369
Week 18	422	423	421	391	425
Week 19	468	480	475	484	482
Week 20	481	480	430	450	473
Week 21	9,348	444	460	489	438
Week 22	4,364	459	463	468	469
Week 23	3,145	461	489	453	515
Week 24	2,722	562	516	495	524
Week 25	2,750	509	549	508	503
Week 26	2,698	467	470	502	488
Week 27	2,754	539	515	526	500
Week 28	2,749	546	536	565	516
Week 29	2,752	511	536	514	549
Week 30	4,346	1,567	3,583	3,225	1,732
Week 31	4,039	1,174	2,872	1,545	1,197
Week 32	5,838	3,515	6,190	7,601	3,936
Week 33	8,947	10,086	15,107	15,100	7,417

Week 34	31,699	5,943	9,644	13,109	16,681
Week 35	20,923	14,743	17,508	18,182	26,809
Week 36	39,371	30,666	32,222	30,491	48,197
Week 37	164,607	56,024	20,747	68,270	30,477
Week 38	101,195	84,606	68,831	95,560	80,384
Week 39	75,439	53,135	23,665	58,825	36,220
Week 40	110,241	88,678	76,132	97,551	89,627
Week 41	77,268	56,246	181,149	64,903	41,558
Week 42	58,808	36,570	104,693	44,767	27,494
Week 43	86,641	62,958	145,833	70,923	77,049
Week 44	111,916	86,403	102,728	97,078	72,737
Week 45	83,047	122,735	76,190	264,090	102,045
Week 46	163,073	144,109	158,139	177,062	130,201
Week 47	157,336	176,631	146,363	138,626	161,407
Week 48	203,214	192,165	191,602	194,442	182,197
Week 49	119,551	140,668	111,969	115,503	129,123
Week 50	10,921	190,908	201,438	190,494	198,326
Week 51	106,938	116,033	96,210	93,396	109,886
Week 52	289,329	222,158	251,260	245,651	222,683
Total	2,076,748	1,911,035	2,057,506	2,119,819	1,810,830

Weekly cost of each run with combination 2 (run 6-10):

Week	Combination 2 Run 6	Combination 2 Run 7	Combination 2 Run 8	Combination 2 Run 9	Combination 2 Run 10
Week 1	388	417	378	377	404
Week 2	394	385	404	395	391
Week 3	398	386	409	418	385
Week 4	408	418	406	390	394
Week 5	440	445	441	444	427
Week 6	407	477	428	435	425
Week 7	436	468	441	460	418
Week 8	462	436	335	453	325
Week 9	568	555	592	570	367
Week 10	433	401	340	385	426
Week 11	516	561	465	515	394
Week 12	413	419	428	435	475
Week 13	514	546	433	543	447
Week 14	410	380	402	361	422

Week 15	388	347	462	495	3,652
Week 16	443	434	503	434	2,748
Week 17	373	416	15,288	400	2,737
Week 18	448	414	7,108	386	2,716
Week 19	481	464	5,606	452	14,792
Week 20	469	488	5,099	472	9,061
Week 21	486	464	4,549	468	7,692
Week 22	466	460	3,996	453	6,867
Week 23	523	491	3,385	495	6,194
Week 24	489	526	3,158	522	5,650
Week 25	527	494	3,172	540	5,576
Week 26	529	536	3,165	522	5,575
Week 27	568	515	3,167	521	5,559
Week 28	496	518	3,171	523	5,562
Week 29	534	531	3,190	546	5,631
Week 30	1,684	2,044	4,995	2,237	6,770
Week 31	1,355	1,309	4,754	1,420	6,518
Week 32	4,987	4,829	6,770	6,125	8,251
Week 33	9,414	9,904	8,898	11,452	11,234
Week 34	17,838	13,710	17,465	18,982	18,227
Week 35	30,518	25,134	31,392	29,073	32,856
Week 36	56,875	49,727	49,869	49,796	58,833
Week 37	37,609	31,528	32,067	30,579	39,317
Week 38	77,167	79,957	81,582	79,819	88,300
Week 39	40,831	37,567	36,296	36,881	43,836
Week 40	86,149	87,579	86,226	86,900	92,580
Week 41	44,678	44,881	37,038	43,103	46,961
Week 42	167,105	30,518	137,805	29,737	173,426
Week 43	109,760	76,572	156,121	72,384	111,352
Week 44	84,886	78,673	114,695	73,106	85,587
Week 45	123,441	108,835	85,661	105,990	127,857
Week 46	147,586	135,123	180,772	128,342	143,428
Week 47	178,174	170,006	171,947	161,800	177,211
Week 48	198,043	187,849	195,017	187,668	191,529
Week 49	150,955	135,102	140,330	129,323	140,462
Week 50	199,640	189,341	214,674	206,561	197,769
Week 51	127,338	113,196	129,750	120,888	119,498
Week 52	255,595	252,961	228,676	224,016	233,452
Total	2,165,034	1,879,739	2,223,722	1,849,593	2,250,966

Weekly cost of each run with combination 4 (run 1-5):

Week	Combination 4 Run 1	Combination 4 Run 2	Combination 4 Run 3	Combination 4 Run 4	Combination 4 Run 5
Week 1	447	427	425	410	429
Week 2	431	440	421	416	424
Week 3	416	468	479	467	442
Week 4	463	444	460	441	468
Week 5	467	453	443	526	482
Week 6	508	500	449	522	452
Week 7	520	477	540	530	523
Week 8	462	563	468	317	527
Week 9	618	601	342	661	601
Week 10	405	330	498	283	316
Week 11	531	532	391	485	560
Week 12	478	522	508	492	466
Week 13	500	484	484	481	512
Week 14	400	432	434	435	409
Week 15	452	460	404	453	495
Week 16	398	390	421	430	393
Week 17	384	354	442	381	397
Week 18	371	408	428	441	373
Week 19	484	491	512	474	489
Week 20	460	477	473	485	491
Week 21	482	440	487	460	430
Week 22	458	475	506	501	490
Week 23	525	476	489	510	489
Week 24	520	527	506	542	540
Week 25	519	534	558	533	528
Week 26	482	487	509	489	456
Week 27	524	528	525	496	568
Week 28	530	509	556	572	501
Week 29	514	547	523	546	509
Week 30	4,052	502	1,849	1,186	1,695
Week 31	2,940	526	1,575	1,009	1,370
Week 32	3,843	1,483	3,424	3,822	2,260
Week 33	5,517	2,587	4,451	6,297	4,195
Week 34	8,540	9,865	14,831	8,952	6,367
Week 35	25,216	97,315	24,817	13,429	10,009

Week 36	40,757	55,406	47,101	22,194	24,062
Week 37	26,514	37,761	28,186	43,890	47,112
Week 38	65,533	66,507	74,196	69,251	75,305
Week 39	28,644	39,474	32,216	39,337	42,992
Week 40	77,189	73,866	82,152	72,835	75,075
Week 41	29,106	39,737	34,015	39,484	44,584
Week 42	53,033	152,208	57,959	22,415	24,973
Week 43	72,220	298,401	78,756	67,047	56,149
Week 44	97,184	226,933	102,538	62,480	69,856
Week 45	62,190	188,113	68,965	89,827	97,435
Week 46	149,772	207,698	149,943	136,571	134,005
Week 47	188,540	161,362	141,693	191,186	208,283
Week 48	163,274	193,315	236,632	183,150	197,069
Week 49	113,332	128,549	132,258	129,445	140,763
Week 50	179,381	183,329	177,547	180,763	176,251
Week 51	90,478	96,324	104,450	98,828	108,463
Week 52	226,160	234,181	224,259	217,554	223,881
Total	1,727,164	2,509,217	1,837,496	1,714,728	1,785,916

Weekly cost of each run with combination 4 (run 6-10):

Week	Combination 4 Run 6	Combination 4 Run 7	Combination 4 Run 8	Combination 4 Run 9	Combination 4 Run 10
Week 1	410	419	417	430	426
Week 2	415	427	411	425	434
Week 3	384	444	314	391	412
Week 4	452	394	434	458	485
Week 5	473	467	456	454	459
Week 6	443	490	431	509	512
Week 7	471	465	474	487	480
Week 8	510	502	552	548	543
Week 9	624	686	692	658	640
Week 10	315	339	324	306	305
Week 11	586	580	634	574	562
Week 12	493	471	510	481	485
Week 13	565	519	494	502	555
Week 14	399	417	435	400	358
Week 15	380	343	341	341	410
Week 16	436	465	448	466	430
Week 17	389	379	371	412	388

Week 18	413	416	401	399	377
Week 19	469	472	499	481	508
Week 20	489	446	485	480	466
Week 21	458	486	459	437	431
Week 22	470	479	489	506	489
Week 23	484	479	465	501	496
Week 24	540	546	510	525	516
Week 25	551	569	528	535	511
Week 26	483	496	531	507	495
Week 27	530	509	513	515	507
Week 28	546	543	544	545	518
Week 29	523	548	540	538	579
Week 30	1,826	7,284	1,140	3,981	527
Week 31	1,763	4,776	1,062	2,013	509
Week 32	2,935	4,438	3,889	18,356	1,944
Week 33	3,473	9,260	5,508	20,036	2,345
Week 34	5,777	54,361	13,635	14,866	6,004
Week 35	25,088	22,690	20,731	22,039	26,991
Week 36	40,929	47,176	37,502	32,329	40,415
Week 37	24,987	28,075	21,859	149,201	24,211
Week 38	66,897	68,891	68,113	89,040	58,881
Week 39	25,041	29,296	24,919	63,279	25,808
Week 40	81,539	78,691	78,441	91,559	76,775
Week 41	27,403	30,656	28,387	60,006	194,183
Week 42	50,337	54,255	51,619	41,304	119,031
Week 43	71,745	74,263	70,333	61,792	301,882
Week 44	94,441	94,860	92,466	79,606	224,922
Week 45	59,879	62,167	264,342	109,468	183,943
Week 46	152,170	152,480	176,275	124,328	203,332
Week 47	143,056	145,348	136,595	139,013	155,306
Week 48	226,398	225,209	175,942	161,891	173,588
Week 49	121,429	118,336	102,233	98,097	117,749
Week 50	181,450	182,445	190,476	172,684	196,539
Week 51	93,429	91,881	83,013	74,647	98,923
Week 52	221,152	225,759	227,718	226,031	243,628
Total	1,736,839	1,826,389	1,889,904	1,869,377	2,491,209

Weekly cost of each run with combination 8 (run 1-5):

Week	Combination 8 Run 1	Combination 8 Run 2	Combination 8 Run 3	Combination 8 Run 4	Combination 8 Run 5
Week 1	362	342	352	357	369
Week 2	381	367	361	352	357
Week 3	438	438	418	442	444
Week 4	431	443	427	416	451
Week 5	395	395	419	404	411
Week 6	498	422	484	419	416
Week 7	464	426	454	451	425
Week 8	468	482	484	431	337
Week 9	428	588	561	572	557
Week 10	454	426	442	441	331
Week 11	466	589	562	569	451
Week 12	426	439	431	468	452
Week 13	490	534	552	490	461
Week 14	459	404	389	406	386
Week 15	464	495	394	432	462
Week 16	414	394	413	418	438
Week 17	400	398	407	382	3,294
Week 18	433	423	417	427	2,347
Week 19	464	452	471	470	1,941
Week 20	510	474	460	487	1,455
Week 21	478	435	458	462	1,263
Week 22	487	481	460	496	1,282
Week 23	449	465	497	474	1,308
Week 24	521	540	499	520	1,351
Week 25	532	498	514	534	1,305
Week 26	537	490	482	497	1,303
Week 27	512	502	493	510	1,348
Week 28	517	537	529	524	1,324
Week 29	542	522	561	507	1,331
Week 30	2,497	2,883	3,004	2,028	2,329
Week 31	2,025	2,631	2,688	1,731	2,046
Week 32	3,987	4,334	3,970	3,384	3,262
Week 33	7,123	5,974	7,608	5,022	5,854
Week 34	8,668	10,097	14,163	8,103	9,711
Week 35	21,060	21,016	25,077	16,264	19,650

Week 36	41,275	39,301	44,234	28,643	39,898
Week 37	107,170	67,471	148,432	58,631	67,704
Week 38	126,851	95,976	95,727	88,877	94,756
Week 39	89,171	64,764	72,398	54,404	62,482
Week 40	125,500	98,573	103,928	86,006	93,868
Week 41	89,554	66,481	73,713	53,520	63,594
Week 42	62,327	44,973	52,885	31,592	42,447
Week 43	88,142	77,190	82,028	56,707	68,587
Week 44	113,049	99,503	102,848	81,240	93,542
Week 45	79,010	128,996	127,860	110,007	121,695
Week 46	163,938	155,465	153,613	133,394	146,197
Week 47	155,493	185,904	181,799	160,482	172,242
Week 48	194,175	197,888	196,057	184,393	183,973
Week 49	186,187	148,348	147,250	123,196	132,703
Week 50	201,916	191,668	192,274	183,163	192,258
Week 51	134,173	121,104	121,094	90,391	107,854
Week 52	211,147	231,970	237,658	227,228	229,701
Total	2,227,859	2,075,910	2,203,699	1,801,760	1,983,953

Weekly cost of each run with combination 8 (run 6-10):

Week	Combination 8 Run 6	Combination 8 Run 7	Combination 8 Run 8	Combination 8 Run 9	Combination 8 Run 10
Week 1	360	339	358	365	377
Week 2	355	372	383	371	373
Week 3	413	448	409	419	437
Week 4	428	396	476	441	437
Week 5	423	408	413	423	417
Week 6	389	423	389	440	406
Week 7	416	443	409	470	426
Week 8	351	438	472	478	459
Week 9	605	538	551	572	591
Week 10	412	418	554	471	414
Week 11	570	539	573	560	578
Week 12	443	438	508	447	422
Week 13	530	538	517	532	552
Week 14	382	372	467	355	387
Week 15	441	534	434	520	473
Week 16	436	396	378	422	422
Week 17	389	396	391	367	410

Week 18	430	422	380	375	402
Week 19	450	481	427	434	437
Week 20	480	468	491	474	512
Week 21	1,684	462	449	451	435
Week 22	1,298	467	448	504	472
Week 23	1,408	509	476	487	483
Week 24	1,417	480	451	512	530
Week 25	1,428	562	534	542	521
Week 26	1,409	500	536	483	503
Week 27	1,432	508	523	503	536
Week 28	1,432	537	511	538	546
Week 29	1,464	574	492	491	517
Week 30	4,494	1,552	2,881	1,608	2,752
Week 31	3,766	1,199	2,592	1,364	2,003
Week 32	6,640	4,376	3,679	3,791	3,471
Week 33	16,887	9,469	6,826	7,401	10,123
Week 34	19,902	15,809	10,531	16,215	15,729
Week 35	29,083	22,787	18,350	30,532	23,735
Week 36	45,786	45,151	37,416	47,561	47,819
Week 37	155,760	72,084	185,235	30,183	28,232
Week 38	98,287	98,480	118,145	83,801	78,423
Week 39	72,010	64,334	86,212	36,840	31,326
Week 40	104,714	94,906	122,659	92,679	85,521
Week 41	70,640	63,824	84,881	39,317	35,300
Week 42	47,985	40,942	55,532	61,948	57,913
Week 43	72,618	70,213	87,284	86,106	83,249
Week 44	96,962	92,053	114,205	110,006	109,317
Week 45	123,719	125,752	80,235	75,563	71,441
Week 46	146,521	149,156	158,844	160,859	157,753
Week 47	176,665	174,999	150,905	143,660	144,310
Week 48	195,488	191,024	201,017	192,045	190,279
Week 49	145,964	140,690	208,380	111,069	232,635
Week 50	193,525	190,131	221,797	200,308	249,634
Week 51	121,688	116,447	146,319	92,427	165,714
Week 52	228,116	235,639	204,931	238,664	192,770
Total	2,198,894	2,034,424	2,322,256	1,877,394	2,032,927

Weekly cost of each run with combination 10 (run 1-5):

Week	Combination 10 Run 1	Combination 10 Run 2	Combination 10 Run 3	Combination 10 Run 4	Combination 10 Run 5
Week 1	376	365	354	361	390
Week 2	376	353	354	366	365
Week 3	412	395	503	446	487
Week 4	525	468	462	494	410
Week 5	378	407	399	386	389
Week 6	407	402	410	396	399
Week 7	450	426	395	396	392
Week 8	327	443	306	320	299
Week 9	351	598	306	461	390
Week 10	318	397	391	394	421
Week 11	444	631	423	359	454
Week 12	477	485	464	405	441
Week 13	441	582	398	448	445
Week 14	422	282	439	376	389
Week 15	2,296	552	2,205	2,617	4,048
Week 16	2,222	387	1,837	1,364	2,244
Week 17	3,514	410	2,529	1,208	1,880
Week 18	3,480	388	2,422	1,232	1,444
Week 19	3,878	475	3,887	1,250	1,236
Week 20	3,889	478	3,860	1,294	1,224
Week 21	4,852	467	5,060	1,284	1,199
Week 22	4,676	465	4,905	1,310	1,211
Week 23	6,063	459	13,021	1,316	1,222
Week 24	6,068	509	10,117	1,319	1,251
Week 25	8,067	535	9,714	1,310	1,273
Week 26	7,948	494	9,031	1,317	1,216
Week 27	8,046	513	8,336	1,337	1,252
Week 28	9,808	517	7,916	1,326	1,269
Week 29	9,555	514	7,892	1,311	1,227
Week 30	10,409	1,209	8,383	1,815	2,612
Week 31	10,313	1,176	8,271	1,749	2,523
Week 32	20,887	3,185	13,293	5,786	3,383
Week 33	19,622	7,456	10,639	5,071	5,428
Week 34	42,668	7,829	34,702	8,002	11,623
Week 35	69,894	14,022	28,164	29,513	42,679

Week 36	90,282	27,201	53,376	53,986	63,564
Week 37	81,851	50,900	59,208	57,064	60,409
Week 38	107,815	78,355	85,748	89,959	86,668
Week 39	87,542	47,374	43,914	35,604	31,674
Week 40	106,438	75,077	88,849	94,113	91,204
Week 41	75,152	46,680	44,554	208,390	80,228
Week 42	50,370	29,301	156,073	137,236	44,548
Week 43	73,047	52,958	146,295	187,734	69,630
Week 44	94,761	65,456	107,318	135,908	93,690
Week 45	141,083	92,171	81,647	103,502	176,122
Week 46	136,221	104,671	134,469	118,630	162,666
Week 47	141,725	103,214	120,455	118,497	162,538
Week 48	160,748	149,220	164,122	156,023	153,681
Week 49	160,452	135,094	148,177	148,711	104,785
Week 50	152,195	130,148	136,975	136,267	168,307
Week 51	90,024	53,544	68,436	79,912	68,646
Week 52	195,224	204,398	213,636	205,367	221,925
Total	2,208,788	1,494,040	2,055,040	2,145,239	1,937,401

Weekly cost of each run with combination 10 (run 1-5):

Week	Combination 10 Run 6	Combination 10 Run 7	Combination 10 Run 8	Combination 10 Run 9	Combination 10 Run 10
Week 1	371	370	373	361	355
Week 2	356	364	372	366	370
Week 3	431	478	436	425	480
Week 4	444	424	399	428	438
Week 5	399	392	392	397	401
Week 6	405	390	411	398	411
Week 7	400	444	404	393	413
Week 8	293	320	355	341	291
Week 9	427	428	452	438	305
Week 10	393	378	368	406	356
Week 11	446	400	464	442	441
Week 12	430	430	456	465	430
Week 13	444	383	437	423	372
Week 14	404	377	395	430	394
Week 15	3,407	2,227	8,116	10,207	1,550
Week 16	1,706	1,751	5,596	4,975	1,352
Week 17	1,309	4,677	7,211	3,646	2,999

Week 18	973	4,027	5,585	3,024	2,727
Week 19	1,029	6,050	7,967	3,066	3,707
Week 20	1,039	4,581	6,233	3,031	4,569
Week 21	15,018	7,243	15,142	13,996	6,730
Week 22	7,275	6,499	9,162	9,217	5,970
Week 23	5,597	5,957	7,610	7,878	6,944
Week 24	4,590	5,374	6,566	7,282	6,197
Week 25	3,729	8,050	5,759	6,855	7,810
Week 26	2,579	7,365	5,278	6,317	7,749
Week 27	2,243	6,779	4,851	6,025	7,738
Week 28	2,184	17,896	4,563	6,019	9,788
Week 29	2,223	14,400	4,549	6,000	9,737
Week 30	4,040	15,224	5,587	7,916	10,616
Week 31	3,770	13,324	5,549	7,859	10,560
Week 32	4,645	15,084	6,746	9,375	27,454
Week 33	8,412	16,296	7,102	9,771	25,883
Week 34	9,937	19,435	23,691	17,075	52,480
Week 35	16,642	38,114	17,286	45,442	31,450
Week 36	35,212	61,949	30,302	64,461	54,100
Week 37	58,793	68,879	52,994	66,727	55,663
Week 38	82,023	99,928	77,177	96,820	94,991
Week 39	51,544	50,861	48,283	205,504	50,655
Week 40	80,956	98,097	80,655	141,564	99,523
Week 41	52,023	51,689	47,502	111,000	51,768
Week 42	32,983	127,066	28,700	76,524	72,027
Week 43	60,932	202,778	62,516	106,818	93,569
Week 44	73,148	150,707	67,167	159,656	157,131
Week 45	128,052	117,997	127,210	114,234	114,477
Week 46	122,066	134,910	121,390	133,345	131,565
Week 47	129,968	132,483	125,842	128,916	133,494
Week 48	152,621	170,110	154,444	166,262	160,083
Week 49	170,002	148,382	140,957	157,305	147,374
Week 50	160,504	139,833	136,818	136,561	135,715
Week 51	87,876	81,798	77,109	76,711	82,136
Week 52	186,944	211,924	203,625	202,464	202,339
Total	1,773,637	2,275,324	1,758,555	2,345,559	2,086,076